M6805UM(AD)

Innovative systems
through silicon.

HMOS

M6805

CMOS

M146805

FAMILY

MICROCOMPUTER / MICROPROCESSOR

USERS MANUAL

MOTOROLA

First Edition

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Chapter 4

Chapter 5

Chapter 6

TABLE OF CONTENTS (Concluded)

Appendices

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1
GENERAL DESCRIPTION

1.0 INTRODUCTION TO THE M6805 FAMILY

The microcomputers and microprocessors of the Motorola M6805 Family are designed to provide an 8-bit processor using a familiar architecture, plus optimization for controller applications. The architecture includes features not usually found on machines of this class such as on-chip timer/counter with interrupt, complete external interrupt, multiple subroutine nesting, true bit manipulation, an index register and numerous configurations.

1.1 PLACE IN THE MICROSPRECTRUM

The M6805 Family architecture and instruction set are very similar to that of Motorola's MC6800. Any programmer who has worked with the MC6800 can attain equivalent proficiency with the M6805 Family in a relatively short time. In some respects the M6805 Family is more powerful than the MC6800 (depending upon the application) as a result of architecture optimization. Appendix A summarizes the architectural and instruction set differences between the M6805 and M6800 Families.

1.1.1 Optimized For Controller Applications

The M6805 Family architecture has been optimized for controller applications, rather than general purpose data processing operations. Several features contribute to this optimization.

The instruction set, used with the M6805 Family, is specifically designed for byte-efficient program storage. Byte efficiency permits a maximum amount of program function to be implemented within a finite amount of on-chip ROM. Improved ROM efficiency allows the M6805 Family to be used in applications where other processors might not perform the task in the available ROM space. More features may be included in applications where ROM space is more than adequate. In some cases the user might wish to include programs for more than one application. In such cases the appropriate program could be selected by the power-up initialization program. The ability to nest subroutines, the addition of true bit test and manipulation instructions, the multi-function instructions, and the versatile addressing modes all contribute to the byte efficiency.

Superficial comparisons of the number of bytes per instruction for the M6805 Family, when compared to other machines in this class, can be very misleading. A single M6805 Family instruction occupying 2 or 3 bytes accomplishes as much real programming work as several single byte instructions, or a subroutine, would accomplish in many other processors.

The bit test and bit manipulation instructions permit the program to:

- branch on bit set
- branch on bit clear
- set bit
- clear bit.

These instructions operate on any individual bit in the first 256 address spaces. As such, the bit manipulations access I/O pins, RAM bits and ROM bits.

One of the chief measures of the effectiveness of a computer architecture is its ability to access data. The M6805 Family has several major memory addressing modes. They include immediate, direct and extended, plus three distinct indexed modes. The programmer is thus given the opportunity to optimize the code to the task. The indexed addressing modes permit conversion tables, jump tables, and data tables to be located anywhere in the address space. The use of look-up tables is an important tool in controller type applications.

Efficient addressing methods are coupled with instructions which manipulate memory without disturbing the program registers. Thus, RAM may be used for the same functions that other processors use general purpose registers (increment, decrement, clear, complement, test, etc.). M6805 Family members have a very versatile, efficient and easy to use I/O structure. All microcomputer I/O function registers are memory mapped into the first 16 address spaces. Advantage is thus taken of the efficient addressing modes, the many memory reference instructions, and the use of RAM (or I/O registers) as general purpose registers. As an example, there are 64 unique instructions which permit the programmer to modify an I/O port. The programmer's problem is not so much how to accomplish a given I/O task, but rather to choose the most effective method from the many methods available. In addition, as with other M6800 Family I/O devices, most M6805 Family I/O pins are individually programmed as inputs or outputs under software control.

1.1.2 M6805 Microcomputer Family Options

A fundamental purpose of the M6805 Family is to offer a common architecture around which various on-chip I/O and memory options are configured. Different microcomputer versions are configured by selecting from among the available options.

The family includes both HMOS (MC6805__) and CMOS (MC146805__) devices, providing a choice as to the technology of the end product. Architectural choices include RAM and ROM size, the number of I/O pins, output drive capability and other kinds of hardware I/O options.

1.2 CHOICE OF TECHNOLOGIES

The first option to be selected by the system designer is the choice between HMOS or CMOS as a processor technology. Appendix B points out the basic difference in HMOS and CMOS technology.

1.2.1 HMOS Feature

The NMOS (N-Channel Metal Oxide on Silicon) technology has been the mainstay of the M6800 Family. The current state of the continual shrinking of NMOS is called HMOS (High-Density NMOS).

The prime consideration in choosing an HMOS M6805 Family microcomputer is its lower price. Motorola's highly-efficient fabrication process results in a greater yield than other processes. The decreased production costs ultimately result in lower selling prices.The economics of large scale production also contribute to a low selling price.

The high speed of Motorola's HMOS, when compared to PMOS or other NMOS processors, produces a very high performance/price ratio.

A low voltage inhibit (LVI) feature may be selected on HMOS versions. The LVI option forces a RESET when the supply voltage drops below a threshold which guarantees correct operation. The CMOS Family members offer wide operating voltage and clock speed ranges, which preclude establishing an LVI threshold.

1.2.2 CMOS Features

An emerging microcomputer technology is CMOS (Complementary MOS, both P- and N-channel devices). The unique properties of CMOS are increasingly attractive. Some applications are simply not feasible with PMOS, NMOS, or HMOS microcomputers.

Maximum power consumption of CMOS parts ranges from 1/15 to 1/200 of that of an equivalent HMOS part. Low power consumption is important in several classes of applications.

(a) **Portable Equipment** — Hand-held and other portable units operate from self-contained batteries. Battery drain is frequently important in such applications; thus, CMOS microcomputers are desirable.

(b) **Battery Back-Up** — CMOS is appropriate in ac powered applications when some or all system functions must continue during a power outage. A small, rechargeable battery keeps a CMOS MCU operable.

(c) **Storage Batteries** — Automotive and telephone equipment operate from larger batteries. Automobile battery drain must be low when the engine is not running. Telephones must operate independent of ac power.

(d) **Heat Dissipation** — Packaging constraints sometimes preclude dissipating electronics generated heat. Or, the heat is costly to dissipate.

(e) **Power Costs** — The cost of electricity to power the equipment becomes a significant factor in calculating the total life cycle cost of equipment which operates continuously.

The CMOS technology inherently operates over a wide range of supply voltages. CMOS is thus appointed where the supply voltage fluctuates, such as in battery powered equipment; or if line power is available, a lower-cost, loosely regulated supply may be used.

An additional advantage of CMOS is that circuitry is fully static. CMOS microcomputers may be operated at any clock rate less than the guaranteed maximum. This feature may be used to conserve power, since power consumption increases with higher clock frequencies. Static operation may also be advantageous during product developments.

1.3 HARDWARE

Every M6805 Family microcomputer contains hardware common to all versions, plus a combination of options unique to a particular version. There are also several differences among family members of which potential users should be aware.

1.3.1 Hardware Common To All Versions

Figure 1-1 details the hardware functional blocks common to all M6805 Family versions.

The central processor unit (CPU) contains the 8-bit arithmetic logic unit, accumulator, program counter, index register, stack pointer, condition code register, instruction decoder, and timing and control logic. These elements resemble the M6800 Family of microprocessors which reflect the M6805 Family heritage.

The M6805 Family has on-chip RAM, permitting the microcomputer versions to operate without external memory. The addressing modes and register-like memory operations use this RAM to the fullest extent possible.

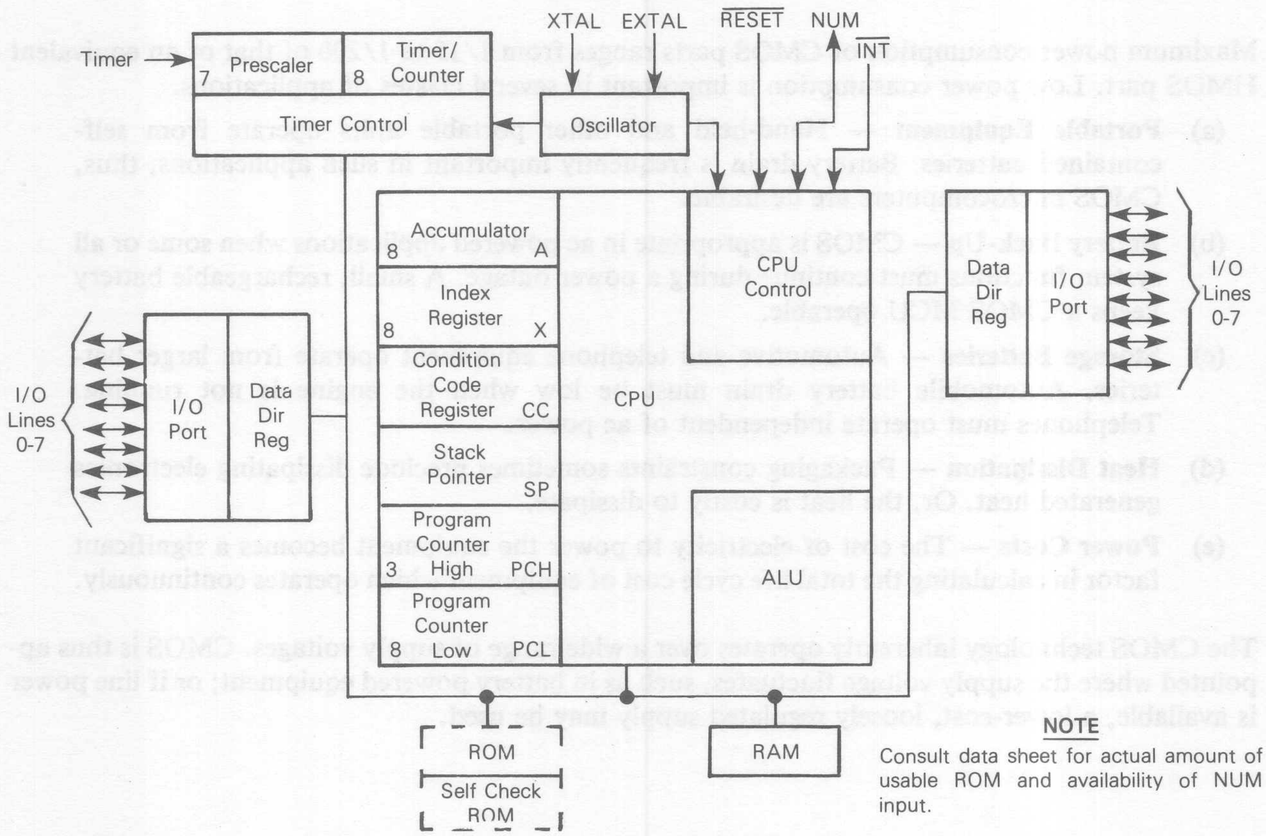


Figure 1-1. MC6805 Family Basic Microcomputer Block Diagram

Parallel I/O capability, with each pin programmable as an input or as an output, is built into every unit.

The external interrupt input, and the capability for multiple nesting of subroutine and interrupts, are features usually found on much more powerful architectures. They permit an M6805 Family MCU to be used in projects usually considered too complex for microcomputers.

A feature which greatly simplifies software development and extends the capability of a microcomputer is an on-chip timer/counter. This 8-bit counter and its prescaler can be programmed for innumerable functions. It can generate an interrupt at software selected intervals. It can also be used as an event counter to generate an interrupt after some software selected number of external events. The timer/counter can also be used for timekeeping, measuring and generating pulses, and counting external events. In the case of the CMOS versions, the timer can be set to "wake-up" the processor from the power-saving WAIT mode.

The external interrupt and timer/counter interrupt are vectored to different service routine addresses. This greatly simplifies interrupt programming. It also speeds execution of interrupt routines, by eliminating software interrupt polling, for determining the source of the interrupt.

The first 16 address spaces are reserved for memory mapped I/O registers. The programmer of the M6805 Family may take full advantage of the versatile addressing modes and the register-like RAM operations of the M6805 Family.

1.3.2 M6805 Family Options

In addition to the common hardware described previously, users can make selections from among devices having a combination of hardware options. Potential users should consult their local Motorola sales representative or the most recent data brochures to determine which versions have reached production.

The first option to be selected by the system designer is the choice of technology. In general, the HMOS units would be selected unless the application specifically requires one of the unique characteristics of CMOS.

User ROM sizes range from none, for the microprocessor, to 2k and larger. Future versions will have additional ROM sizes. When self-check ROM is a part of the device, the ROM area used in the self-check operation is not included in the published ROM sizes. The user gets the entire ROM space for his program.

A small portion of the ROM is located in page zero (the direct page) to facilitate more efficient access to lookup tables using all available addressing modes. This ROM can, of course, be used for program storage as well as lookup tables.

The initial M6805 Family versions contain either 64 or 112 bytes of on-chip RAM which is located in page zero. Future versions will accomodate additional or differing amounts of RAM.

Package size options permit as many as four, full 8-bit bidirectional I/O ports. Each pin is defined under software control as an input or output by loading a data direction register.

Electrical options include TTL compatibility, CMOS compatibility, and high-current outputs designed to drive darlington transistors and LEDs.

Complex I/O functions are also included in selected versions of the M6805 Family. For example, an on-chip, high-speed, successive approximation type, 8-bit, analog-to-digital converter is included on one early member of the family.

The expandable CMOS microprocessor version uses a multiplexed address-then-data bus. The expandable version is used with related peripheral and memory ICs to implement larger systems. Prototyping ROM-based microcomputers is a second use of the expandable version.

Zero-crossing detection circuitry, which is connected internally to the external interrupt-input pin of some versions, can be interfaced with a power line or other source of periodic input for time-keeping functions. It can also be used by the program to synchronize outputs to the zero-crossing of the power line voltage.

1.3.3 Differences Between Family Versions

There are some significant differences among the products being offered which might be of concern to the system designer.

Pinouts — M6805 Family members having similar features might not have identical pinouts, due to manufacturing factors taken into consideration during design. This should not cause problems for users since the decision of which version to use is made early in the design cycle. A switch from one version or one technology to the other is unlikely.

STOP and WAIT Instructions — In order to further decrease the power consumption of the CMOS versions, two instructions (STOP and WAIT) are added.The STOP and WAIT instructions disable the clock signal to all or portions of the internal logic. This eliminates dynamic power dissipation which accounts for most of the power used in a CMOS microcomputer. The clock is reenabled when the timer counter reaches zero and/or an external interrupt is received.

Fewer Cycles — All family versions execute the same instructions (except for STOP and WAIT). But some versions of the M6805 Family require fewer clock cycles to execute the instructions. Most programs are not affected by the difference. Since a hardware timer is included, software timing intervals are not often needed. Individual data sheets for each family member list the number of clock cycles required to execute each instruction. The fastest M6805 Family members execute code at a speed equal to that of the M6800 Family for those instructions which are directly comparable.

Clock Divider — Most versions use a divide-by-four on the clock input to generate the internal bus timing. The microprocessor version requires more resolution to generate the bus interface and control signals. Thus it uses a divide-by-5 clock input to generate the internal bus timing.

Software Configurable Timer — Not all of the microcomputer versions permit the programmer to configure the timer/counter, prescaler and clock source under software control. In some this is done in hardware, using the ROM mask layer to define timer/counter operation.

The most reliable method of obtaining specific details, for a particular version of the M6805 Family, is to consult the most recent data sheet describing the version of interest. These data sheets and other literature are available from your local Motorola sales representative or franchised distributor.

1.4 ENHANCED MICROCOMPUTER TEST CAPABILITY

As the complexity of VLSI (Very Large Scale Integration) rises, increasingly complex and costly test hardware is required. This is especially true of ROM-based microcomputers. Implementation of the user's program in ROM essentially creates a custom part for every customer program.

An added cost for the user is that of incoming inspection testing.

M6805 Family microcomputers have a sophisticated on-chip self-check capability. This consists of a ROM area, separate from the user's ROM, which contains a program designed to exercise the majority of on-chip hardware. The self-check ROM is accessed only when the microcomputer is placed in the self-check mode.

The self-check program is designed to exercise the on-chip circuitry to ensure that it is operable. The test program includes software which checks the RAM, ROM, I/O ports, external interrupt, and the timer. Although it cannot check execution of every possible instruction, it is designed to exercise the vast majority of on-chip logic.

The self-check program requires user assembly of only a socket and a few inexpensive components (costing approximately ten dollars). The assembled tester is contrasted to the most advanced and expensive integrated circuit testers used by Motorola for the factory final test. Figure 1-2 shows a schematic diagram of the typical connections required to test the MC6805P2 Microcomputer member of the M6805 Family. All four LED indicators will flash; however, the LED connected to pin 11 flashes at about a 3 Hz rate. If the pin 11 LED indicator does not flash, the unit is defective. Other members of the family require similar connections to match their specific I/O configuration.

1.5 MICROPROCESSOR SYSTEM IN CMOS

The MC146805E2 Microprocessor is designed as a general-purpose CMOS microprocessor for applications requiring a multi-chip CMOS system. It also serves as a development tool for ROM-based microcomputer versions. It is supported by a line of CMOS memories, peripherals and high-speed logic to simplify system design.

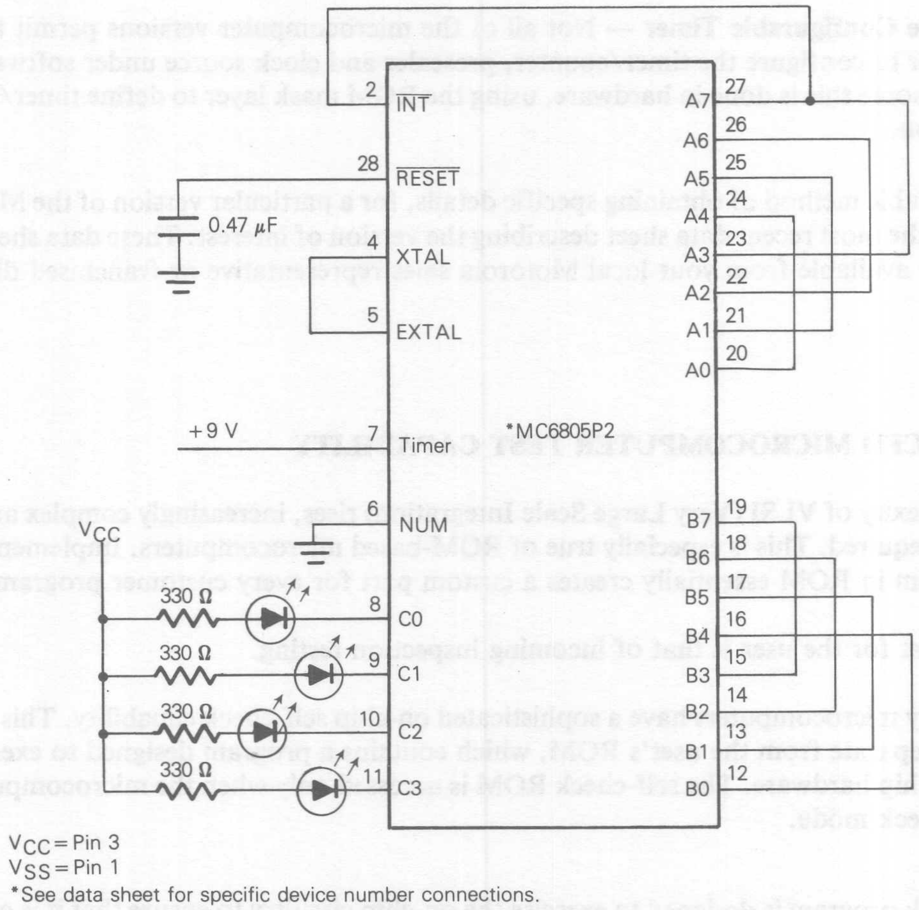


Figure 1-2. Example of Self-Check Schematic Diagram for MC6805P2

1.5.1 MC146805E2 Microprocessor

The 40-pin microprocessor contains the processor, 112 bytes of RAM, an expansion bus, and 16 I/O lines. The eight low-order address bits and eight data bits are time-multiplexed on the bus pins. Multiplexing of the bus is controlled by three bus control lines. Five additional non-multiplexed address lines permit a total address space of 8k bytes. Bank-switching techniques could be used to extend this address space as required. System interface problems are reduced by including bus driver outputs on all bus pins.

1.5.2 Peripherals

Any microprocessor-based system is heavily dependant upon easily interfaced peripherals for cost-effective system design. The MCM65512 RAM, MCM65516 ROM, MC146823 Peripheral Interface Adapter, MC146824 Peripheral Interface Adapter plus Timer, and MC146818 Real-Time Clock plus RAM all include the following design simplifying characteristics:

1) **Bus Drivers** — All bus interface pins are designed to drive a capacitive load of 130 pF at maximum clock frequency. The use of off-chip bus drivers is thus eliminated in many systems.

2) **Bus Compatablity** — The peripherals and memories are designed to operate directly on Motorola MC146805 and MC6801 multiplexed buses. Other type multiplexed buses (8085/8048/8086, etc.) are also easily accomodated by the CMOS peripheral and memory circuits.

1.5.3 High-Speed Bus Logic

On complex microprocessor-based systems, a family of high-speed logic is required to perform functions such as driving bus loads, address decoding, bus control functions, etc. Table 1-1 outlines some of the high-speed CMOS devices designed by Motorola to implement these functions.

Table 1-1. High-Speed CMOS Bus Logic

Octal, Bus Driver
Octal, Three-State Bus Transceiver
3-to-8 Latched Decoder
3-to-8 Decoder
Octal, Three-State Transparent Latch
Hex Inverter
Quad 2-Input NAND Gate
8-Input NAND Gate
Dual D Flip-Flop

1.6 SOFTWARE DEVELOPMENT

Microcomputers accomplish a task by using software to define the operations of the microcomputer at the programming stage, rather than by using digital logic to construct a system which has its operations defined in the design stage. Therefore, software development serves a function similar to logic design. Appendix C provides information concerning the assembly language syntax and assembler directive for the M6805 Family.

1.6.1 Critical Factor for Product Success

Since software development replaces most of the digital design function in microcomputer systems, it is important that error-free software be generated. Because many of the microcomputers are used in projects which exist in highly competitive markets, it is also essential that the error-free software be ready for product introduction during the most advantageous "product window marketing time."

To produce error-free software at the right time, a thorough development and debug system is a necessity. Also, since software development usually consumes most of the project start-up costs, an efficient development system makes the programmer's task less difficult, thus paying for itself by the time saved.

1.6.2 Software Development

Table 1-2 lists the various phases of a software development process together with the Motorola Support Products used in each phase.

Table 1-2. Software Development Phase

Software Development Phase	Motorola Support Products
Evaluation	M6805 Evaluation Module with Debug
Code Generation	EXORciser with MDOS Operating System and Macro Assembler
Debug	EXORciser with MDOS and MEX6805 with Software Package
Prototype	MC146805E2 Microprocessor with MCM2716 UV EPROM
Mask Programmed Device Verification	EXORciser with MDOS and MEX6805 with Software Package

The first phase in the software development process takes place prior to selection of the actual microcomputer to be used. This stage includes evaluation of various microprocessors and microcomputers to determine the one best suited for the particular system. When evaluating the M6805 Family, an M6805 Evaluation Module with Debug can be used. This module, available through local Motorola Sales Offices, provides the opportunity to gain "hands-on" experience with the M6805 Family during this phase.

Once the M6805 Family part is selected for the system, the Code Generation phase begins. The efficient instruction set of the M6805 Family, together with the convenience of the EXORciser-based development system and the macro assembler, all contribute to programmer efficiency during the code generation phase.

When the code is written and assembled, the EXORciser-based software development board is connected to the rest of the microcomputer system. This allows simultaneous checkout of both the software and the hardware under control of the development system supervisory software.

Once the software and hardware are functioning as expected, prototype systems can be constructed for field trials, using EPROM devices. The field-trial phase of prototype development can uncover hidden bugs since the equipment might be field operated in ways which are difficult to forsee during the design phase. Field trials also provide user feedback which could result in beneficial changes to the final version of the product. EPROM versions of the microcomputer itself or a microprocessor version using separate EPROMs are two methods of constructing prototype systems.

The final steps in the software development include submittal of the code to Motorola and verification of correct microcomputer system operation by using mask programmed samples, supplied by Motorola, prior to volume production.

1.6.3 Unified Development System

Since the software development and debug phase of product development is so important to the success of a microcomputer-based product, Motorola provides the hardware and software necessary to ensure that software development can be accomplished efficiently and thoroughly. The M6805 Support System is used, together with the Motorola EXORciser and MDOS operating system, to help debug the M6805 Family. Figure 1-3 provides a block diagram of the support system used in the software checkout.

The M6805 Support System (part number MEX6805) consists of a circuit board(s), cables with connectors, and an MDOS diskette. The support system circuit board(s) contains either an HMOS or a CMOS M6805 Family processor which executes code in real time from either on-board RAM or on-board EPROM. The circuit board(s) also contains breakpoint hardware and a wiring area for

system modification. A DIP connector is also provided for direct connection to the microcomputer socket of the end product. Thus, the support system permits both the hardware and the software to be debugged as an assembly rather than individually. This also helps to locate interface errors which could otherwise be difficult to isolate.

The software package supplied, with the support system, provides the programmer wtih direct control of the M6805 Family processor. Programs can be loaded from the disk drive, breakpoints can be set and program operation can be traced directly from the EXORciser terminal. These and other features reduce the debug phase of both the hardware and the software to a minimum and provide a high level of confidence that the code has been debugged.

Motorola's macro assembler and linking loader give the M6805 Family software development system capabilities usually available only for the most sophisticated microprocessors.

Appendix C contains a description of the macro assembler directives. It includes a versatile macro capability, conditional assembly and numerous other features which simplify the programmer's task Other versions of the same basic macro assembler are able to assemble programs written for other Motorola processors such as the M6800, M6801 and M6809.

The EXORciser is, of course, a general purpose microcomputer system which supports several high level languages and can be used for tasks other than software development. It need not sit idle between software development projects.

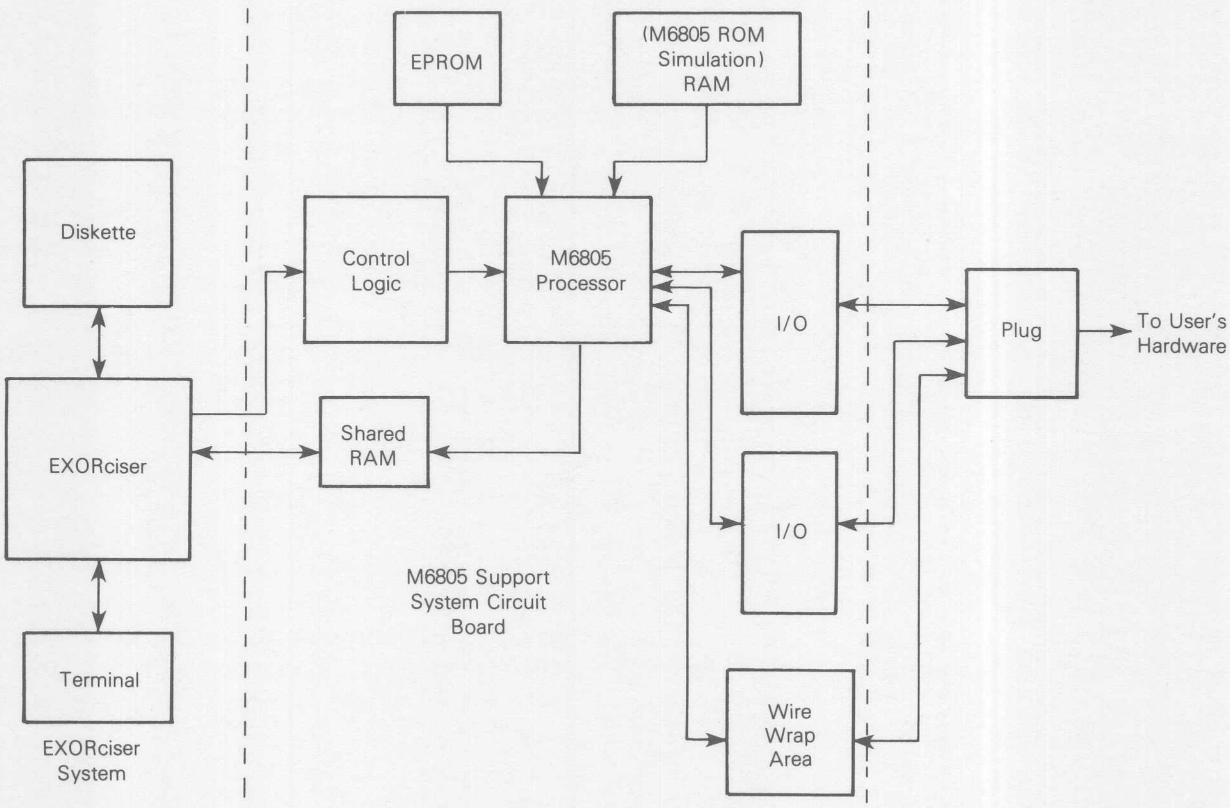


Figure 1-3. M6805 Support System Block Diagram

CHAPTER 2
PROGRAMMING FEATURES

Software implementation for the M6805 Family closely follows the MC6800 heritage. Since the programming features are similar, many of the MC6800 programming features are inherent to the M6805 Family. Some key M6805 Family features are listed below:

- ROM Byte Efficient
- Easy to Program
- Versatile Interrupt Handling
- True Bit Manipulation
- Bit Test and Branch Instructions
- Powerful Addressing Modes
- Consistent Instruction Set
- Indexed Addressing for Table Lookup
- Powerful Instruction Set
  - All MC6800 Arithmetic Instructions
  - All MC6800 Logical Instructions
  - All MC6800 Shift Instructions
  - Full Set of Conditional Branches

CHAPTER 3
ARCHITECTURE

3.0 PROCESSOR ARCHITECTURE

3.1 M6805 FAMILY PROGRAMMING MODEL

The M6805 Family processor contains five registers. The accumulator (A) and index register (X) are 8-bit registers, while the condition code (CC) register contains five bits. The program counter (PC) and stack pointer (S) vary in length, depending upon the version of the family. The PC of initial versions is 11, 12, or 13 bits long, depending upon memory size. As far as accessing memory is concerned, S is the same length as PC. However, the high order bits are fixed. The initial versions have either five or six register bits in S, depending upon the size of on-chip RAM. The M6805 Family Register Architecture is shown below.

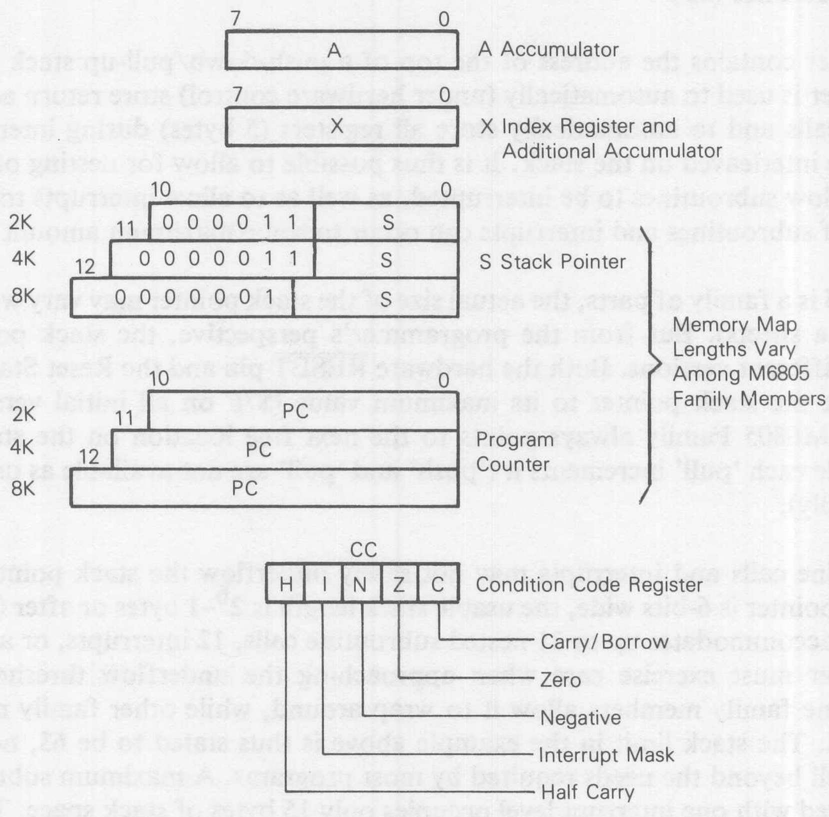


Figure 3-1. M6805 Family Register Architecture

3.2 ACCUMULATOR (A)

The A-register is a general purpose accumulator that is used by the program for arithmetic calculations and data manipulations.

3.3 INDEX REGISTER (X)

The X-register is used during the indexed modes of addressing, as well as an auxiliary accumulator. In indexed instructions, the X-register provides an 8-bit value that is added to an optional instruction-provided value, to create an effective address. For more information see the section on Addressing Modes. The X-register is also used on the M6805 Family for limited calculations and data manipulation. The full set of read/modify/write instructions operate on the X-register, as well as the accumulator. Code sequences which do not employ the index register for indexed addressing may use X as a temporary storage cell, or accumulator.

3.4 PROGRAM COUNTER (PC)

The program counter is used by the processor to point to the next instruction to be executed by the processor. Though the PC on early M6805s is 11, 12, or 13 bits, the family architecture supports a PC of up to 16 bits.

3.5 STACK POINTER (SP)

The stack pointer contains the address of the top of a push-down/pull-up stack located in RAM. The stack pointer is used to automatically (under hardware control) store return addresses (2 bytes) on subroutine calls and to automatically store all registers (5 bytes) during interrupts. The saved registers may be interleaved on the stack. It is thus possible to allow for nesting of subroutines and interrupts, to allow subroutines to be interrupted, as well as to allow interrupts to call subroutines. This 'nesting' of subroutines and interrupts can occur to some maximum amount described below.

Since the M6805 is a family of parts, the actual size of the stack pointer may vary with RAM size (see appropriate data sheets). But from the programmer's perspective, the stack pointers all appear similar on the different versions. Both the hardware $\overline{\text{RESET}}$ pin and the Reset Stack Pointer (RSP) instruction reset the stack pointer to its maximum value ($7F on all initial versions). The stack pointer on the M6805 Family always points to the next free location on the stack. Each 'push' decrements while each 'pull' increments it ('push' and 'pull' are not available as user instructions in the M6805 Family).

Nested subroutine calls and interrupts may not safely underflow the stack pointer. For example, when the stack pointer is 6-bits wide, the usable stack length is 2^6-1 bytes or after 63 bytes are pushed. The 6-bit S accommodates up to 31 nested subroutine calls, 12 interrupts, or a mixture of both. The programmer must exercise care when approaching the underflow threshold. When the S underflows, some family members allow it to wrap around, while other family members produce different results. The stack limit in the example above is thus stated to be 63, not 64, bytes. The stack limit is well beyond the needs required by most programs. A maximum subroutine nesting of five levels coupled with one interrupt level occupies only 15 bytes of stack space. The allowed stack length is typically traded off against the needed data RAM space.

3.6 CONDITION CODE REGISTER (CC)

The condition code register contains various flag bits that reflect the current state of the processor. Most CC bits reflect the results of the last executed data reference instruction. The effect of each instruction on the CC bits is listed with the instructions in Section 8. These CC bits are described briefly below.

3.6.1 Half Carry (H)

The H-bit is set when a carry occurs between bits three and four during an ADD or ADC instruction. The half-carry flag may be used in BCD addition subroutines.

3.6.2 Interrupt Mask (I)

When the I-bit is set, the external interrupt and timer interrupt are masked (disabled). Clearing the I-bit allows interrupts to be enabled. If an Interrupt occurs while the I-bit is set, the interrupt is latched and causes the interrupt vector to be fetched when the I-bit is next cleared.

3.6.3 Negative Bit (N)

The N-bit is set when bit seven of the result of the last data manipulation, arithmetic, or logical operation was set. This indicates that the result of the operation was negative.

3.6.4 Zero Bit (Z)

The Z-bit is set if the result of the last data manipulation, arithmetic, or logical operation was zero.

3.6.5 Carry Bit (C)

The C-bit is set if a carry or borrow out of the 8-bit ALU occurred during the last arithmetic operation. Also, the C-bit is set during shift, rotate, and bit test instructions.

CHAPTER 4
ADDRESSING MODES

4.0 INTRODUCTION

The power of any computer (either large or small) lies in its ability to access memory. The addressing modes of the processor provide that capability. The M6805 Family has a set of addressing modes that meets these criteria extremely well, especially for a processor in its price range.

In the following descriptions the term effective address (EA) is used. The EA is the address in memory from which the argument for an instruction is fetched or stored. In two operand instructions, such as add to accumulator (ADD), one of the effective operands (the accumulator) is inherent and not considered an addressing mode per se.

Descriptions and examples of the various modes of addressing the M6805 Family are provided in the paragraphs which follow. Several program assembly examples are shown for each mode, and one of the examples is described in detail (ORG, EQU, and FCB are assembler instructions and not an instruction set mnemonic). Parenthesis are used in these descriptions/examples, of the various addressing modes, to indicate "the contents of" the location or register referred to; e.g., (PC) indicates the contents of the location pointed to by the PC. The colon symbol (:) indicates a concatenation of bytes. In the following examples, the program counter (PC) is initially assumed to be pointing to the location of the first opcode byte. The first PC + 1 is the first incremental result and shows that the PC is pointing to the location immediately following the first opcode byte.

4.1 IMMEDIATE ADDRESSING MODE

The EA of an immediate mode instruction is the location following the opcode. This mode is used to hold a value which is known at the time the program is written, and which is not changed during program execution. These are two byte instructions, one for the opcode and one for the immediate data byte.

$$PC + 1 \rightarrow PC$$
$$EA = PC$$
$$PC + 1 \rightarrow PC$$

Assembly Examples:

```
0400 A6 03        A        LDA     #$03
0402 AE C3        A        LDX     #$C3
0404 A3 FF        A        CPX     #$FF
05BE                       ORG     $5BE
05BE A6 F8        A        LDA     #$F8    (See example description below.)
```

Figure 4-1 shows an example of the Immediate Addressing Mode. In this example, the program contains an instruction to load the accumulator with the hexadecimal number F8, which is the byte immediately following the opcode byte.

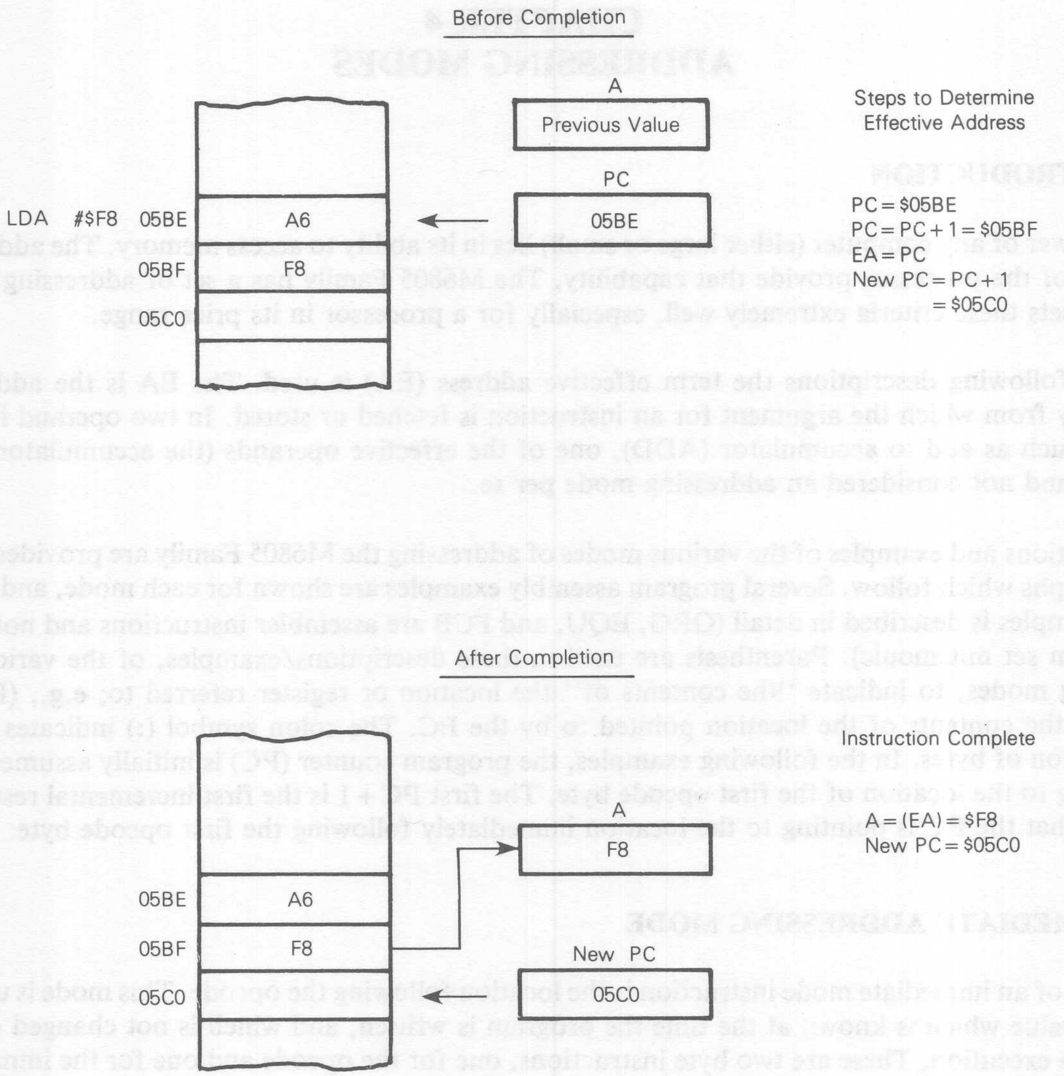


Figure 4-1. Immediate Addressing Mode Example

4.2 DIRECT ADDRESSING MODE

The EA of a direct mode (DIR) instruction is the contents of the next byte of the opcode. Direct addressing can be used to reference any of the first 256 ($00-$FF) locations of memory with a two byte instruction. In the M6805 Family, direct addressing can be used to reference all I/O and RAM locations as well as some ROM. This is a two byte instruction.

$$PC + 1 \rightarrow PC$$
$$EA = (PC) + \$0000$$
$$PC + 1 \rightarrow PC$$

Assembly Examples:

```
0400 B6 50         A          LDA     $50
        0030       A  DOG     EQU     $30
0402 BE 30         A          LDX     DOG
0404 3C 27         A          INC     $27
0406 12 30         A          BSET    1,DOG
052D                          ORG     $52D
        004B       A  CAT     EQU     $4B
052D B6 4B         A          LDA     CAT
```

(See example description below.)

Figure 4-2 shows an example of the Direct Addressing Mode. In this example, the program contains an instruction to load the accumulator with CAT. (CAT in this example is equal to the contents of memory location 004B, which is the result of adding the byte following the opcode byte to $0000.)

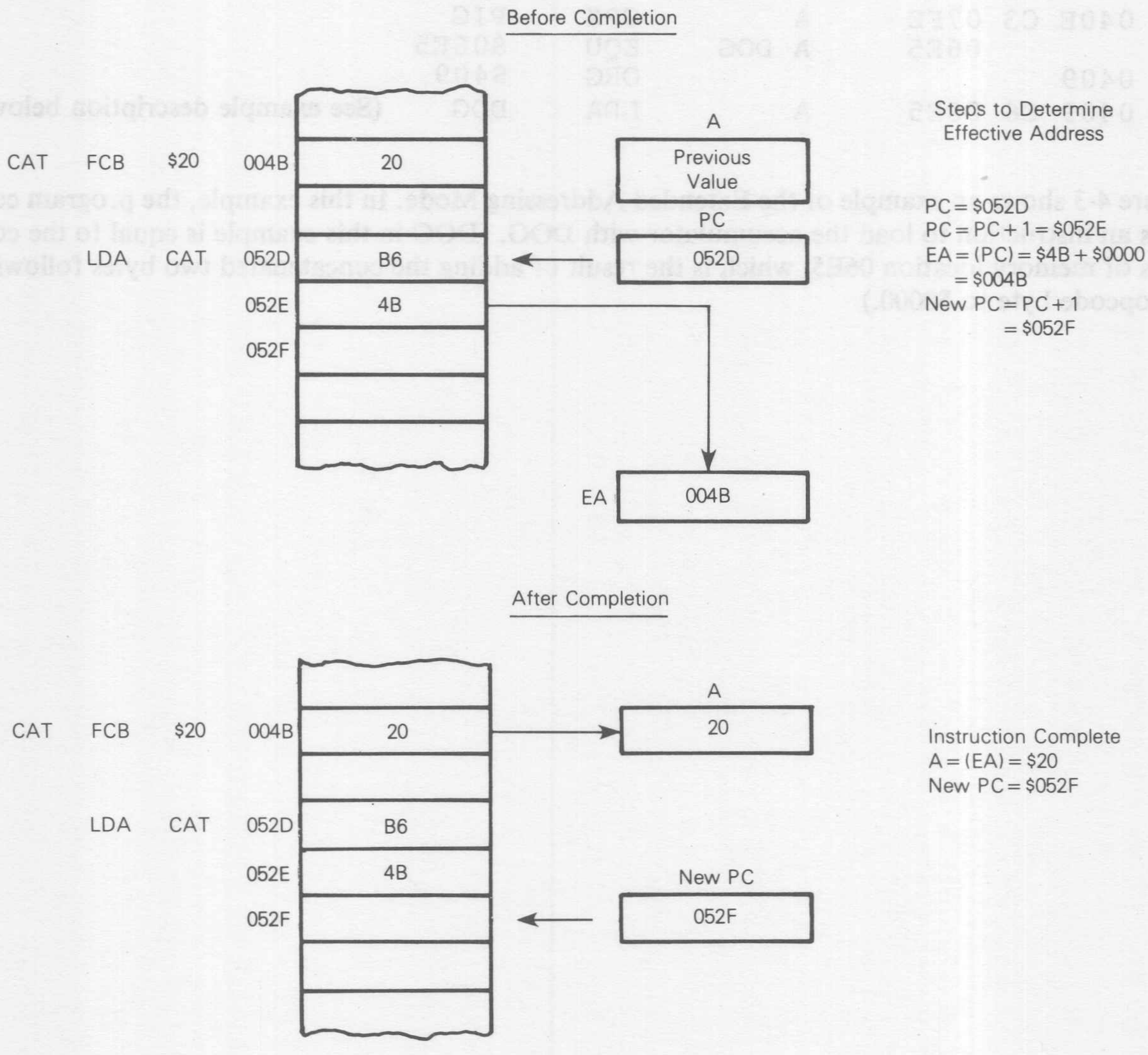


Figure 4-2. Direct Addressing Mode Example

4.3 EXTENDED ADDRESSING MODE

The EA of an extended mode instruction is the contents of the two bytes following the opcode. Extended addressing references any location in the MC6805 memory space, I/O, RAM and ROM. Also, since the two bytes following the opcode contain 16 bits, the addressing range of the M6805 Family may be extended in the future without affecting the instruction set or addressing modes. Extended addressing mode instructions are three bytes long, the one byte opcode plus a two byte address.

$$PC + 1 \rightarrow PC$$
$$EA = (PC) : (PC + 1)$$
$$PC + 2 \rightarrow PC$$

Assembly Examples:

```
          07FE     A PIG    EQU    $7FE
040E C3   07FE     A        CPX    PIG
          06E5     A DOG    EQU    $06E5
0409                        ORG    $409
0409 C6   06E5     A        LDA    DOG
```
(See example description below.)

Figure 4-3 shows an example of the Extended Addressing Mode. In this example, the program contains an instruction to load the accumulator with DOG. (DOG in this example is equal to the contents of memory location 06E5, which is the result of adding the concatenated two bytes following the opcode byte to $0000.)

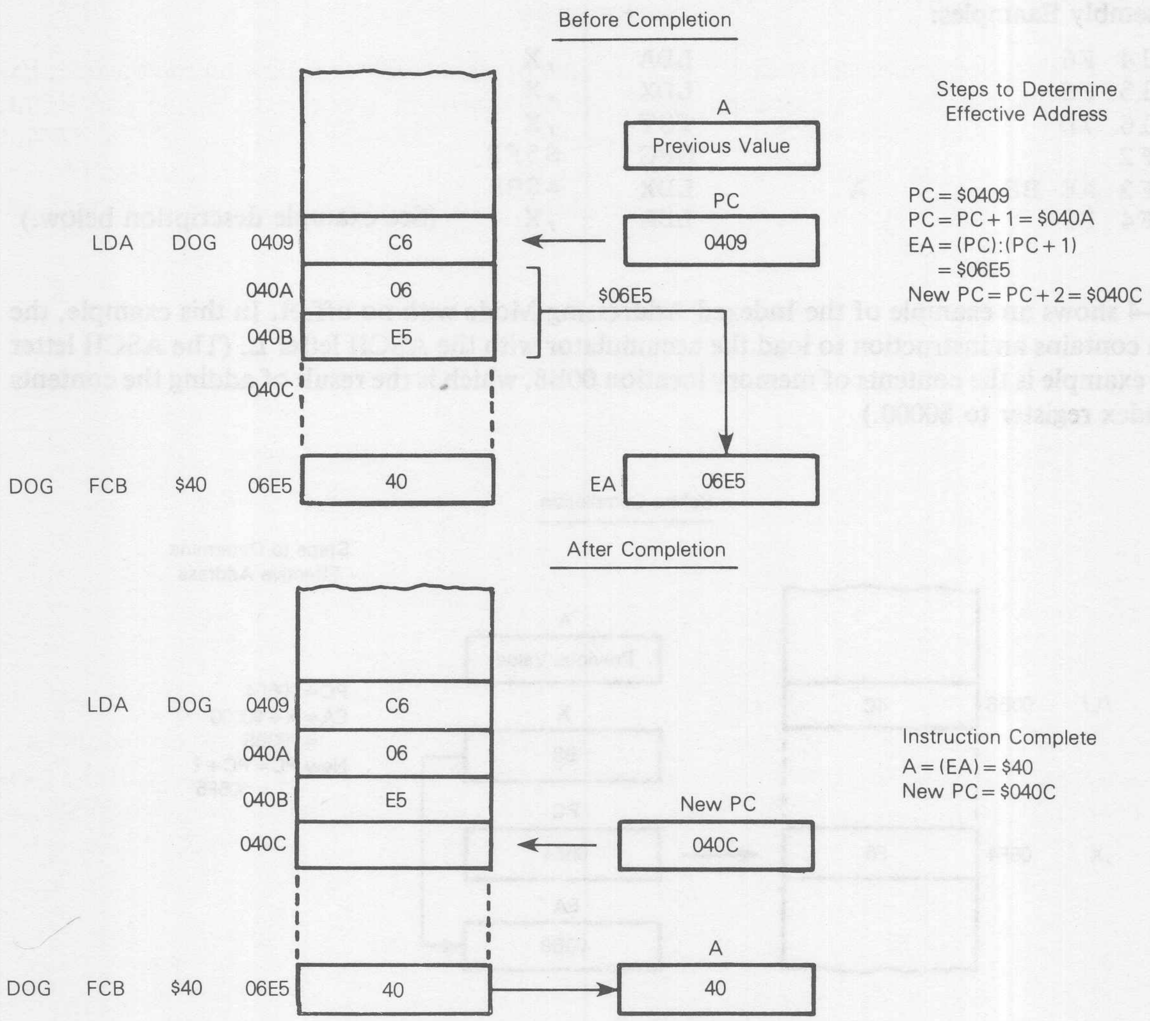


Figure 4-3. Extended Addressing Mode Example

4.4 INDEXED ADDRESSING MODE

In the indexed addressing modes the X-register (index register) is used in the calculation of the EA. Three types of indexed addressing exist in the M6805 Family.

4.4.1 Indexed — No Offset

In this mode the contents of the index register is the EA. This mode is used to create EAs pointing to data in the lowest 256 bytes of the address space, including I/O, RAM and part of ROM. It may be used to move a pointer through a table, point to a frequently referenced location (e.g. — an I/O location), or hold the address of a piece of data that is calculated by a program. Indexed, no offset, instructions use only one byte, the opcode.

$$EA = X + \$0000$$
$$PC + 1 \rightarrow PC$$

Assembly Examples:

```
0414 F6                LDA    ,X
0415 FE                LDX    ,X
0416 7D                TST    ,X
05F2                   ORG    $5F2
05F2 AE B8         A   LDX    #$B8
05F4 F6                LDA    ,X
```

(See example description below.)

Figure 4-4 shows an example of the Indexed Addressing Mode with no offset. In this example, the program contains an instruction to load the accumulator with the ASCII letter L. (The ASCII letter L in this example is the contents of memory location 00B8, which is the result of adding the contents of the index register to $0000.)

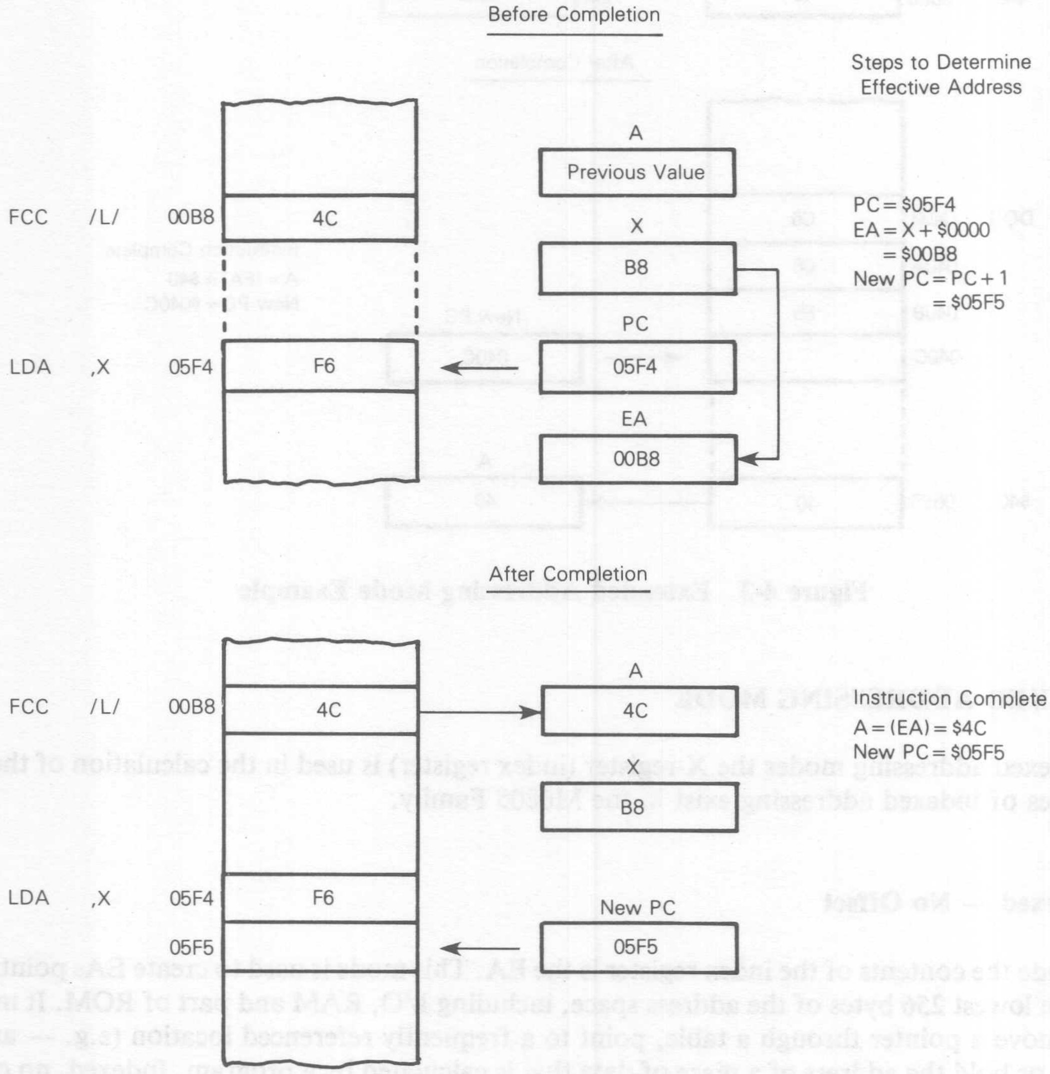


Figure 4-4. Indexed Addressing Mode, No-Offset Example

4.4.2 Indexed — 8-Bit Offset

The EA is calculated by adding the contents of the byte following the opcode to the contents of the index register. This mode is useful in selecting the kth element in an n element table. To use this mode the table must begin in the lowest 256 memory locations, but may extend through the first 511 memory locations of the M6805 Family. All indexed 8-bit offset addressing can be used for ROM, RAM or I/O. This is a two byte instruction, the opcode byte and the offset byte. ROM efficiency encourages the inclusion of as many tables as possible in page zero and page 1.

$$PC + 1 \rightarrow PC$$
$$EA = (PC) + X + \$0000$$
$$PC + 1 \rightarrow PC$$

Assembly Examples:

```
          0040     A LIST    EQU    $40
0417 E6 40         A         LDA    LIST,X
0419 6C 40         A         INC    LIST,X
          0089     A TABLE   EQU    $0089
0759                         ORG    $759
0759 AE 03         A         LDX    #$03
075B E6 89         A         LDA    TABLE,X (See example description below.)
```

Figure 4-5 shows an example of the Indexed Addressing Mode with 8-bit offset. In this example, the program contains an instruction to load the accumulator with a tabular value containing the hexadecimal number \$CF. (\$CF in this case is contained in memory location \$008C, which is the result of adding the byte following the opcode to the contents of the index register plus \$0000.)

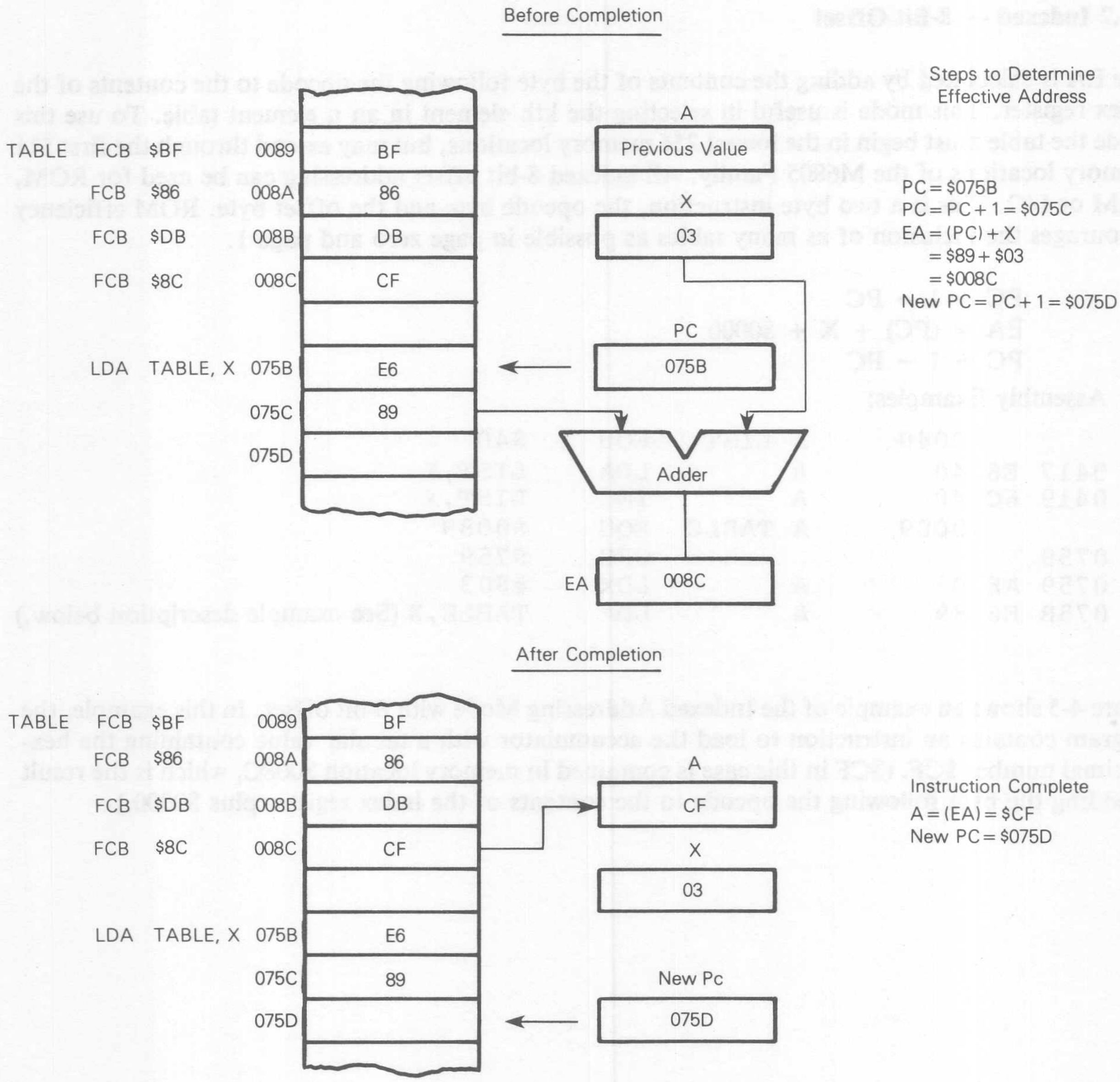


Figure 4-5. Indexed Addressing Mode, 8-Bit Offset Example

4.4.3 Indexed — 16-Bit Offset

The EA for the 2-byte offset mode is calculated by adding the concatenated contents of the next two bytes following the opcode to the contents of the index register. This mode is used in a similar manner to indexed with one byte offset; except that since the offset is 16 bits, the tables being referenced can be anywhere in the memory space. For more details see the Compatibility paragraph below. This is a three byte instruction, one for the opcode and two for the offset value.

$$PC + 1 \rightarrow PC$$
$$EA = (PC) : (PC + 1) + X$$
$$PC + 2 \rightarrow PC$$

Assembly Examples:

```
               0700     A COW      EQU     $700
041B  D6      0700     A          LDA     COW,X
041E  DA      0700     A          ORA     COW,X
               077E     A TABL     EQU     $77E
0690                               ORG     $690
0690  BE      02       A          LDX     $02
0692  D6      077E     A          LDA     TABL,X   (See example description below.)
```

Figure 4-6 shows an example of the Indexed Addressing Mode with 16-bit offset. In this example, the program contains an instruction to load the accumulator with a tabular value containing the hexadecimal number $DB. ($DB in this case is contained in memory location 0780, which is the result of adding the concatenated two bytes following the opcode byte to the contents of the index register.)

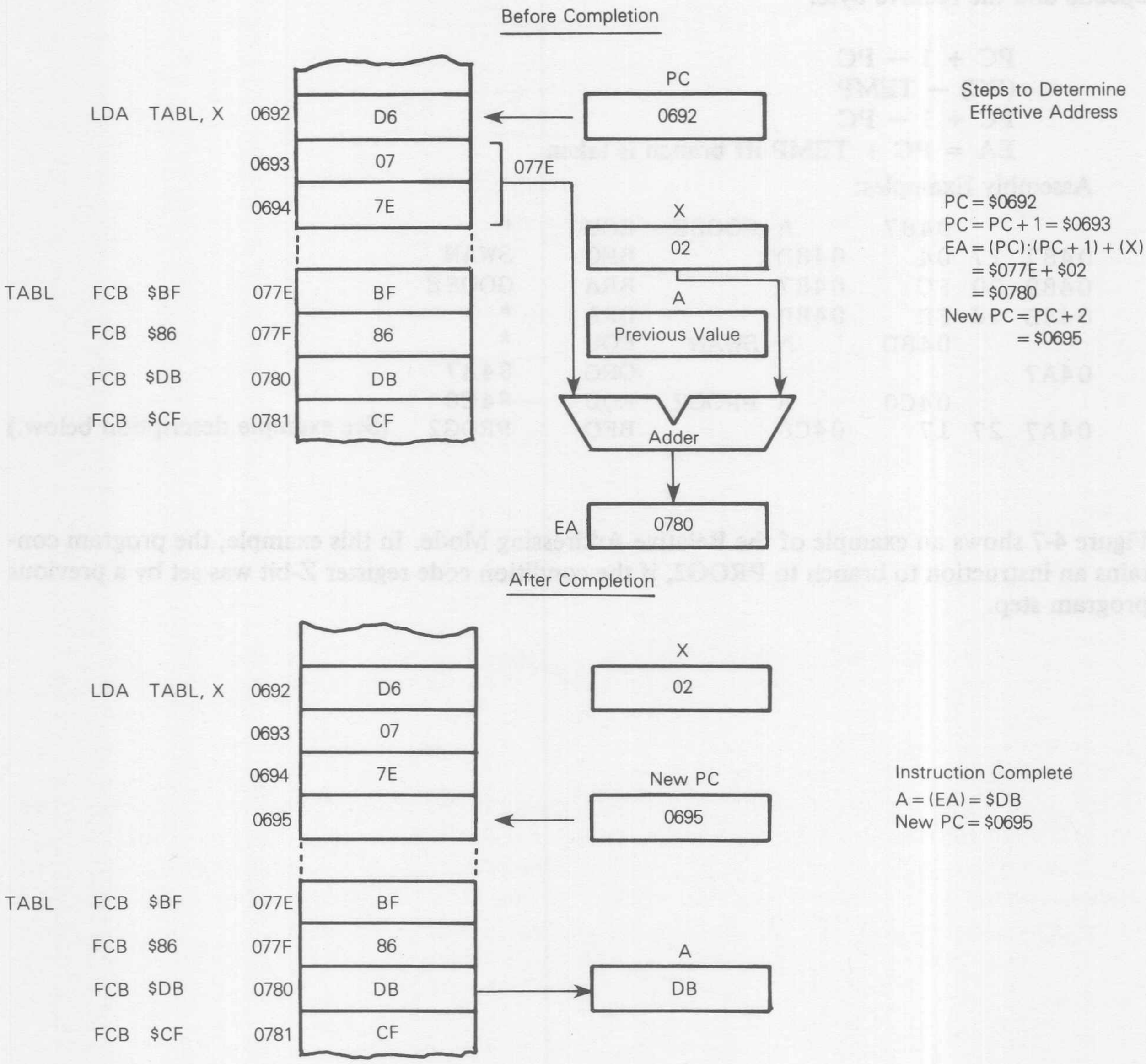


Figure 4-6. Indexed Addressing Mode, 16-Bit Offset Example

4.4.4 Indexing Compatibility

Since the index register on the M6805 Family is only eight bits long, and the offset values are 0, 8, or 16 bits, the MC6800 user may thus find that the X-register on the M6805 Family is best utilized 'backwards' from the MC6800. That is, the offset will contain an address or pointer to the table and the index register contains the displacement into the table.

4.5 RELATIVE ADDRESSING

Relative (REL) addressing adds the contents of the byte following the opcode to the value of the program counter (PC). The resultant EA is used if, and only if, a relative branch is taken. Note that by the time the byte following the opcode is added to the PC, the PC is already pointing to the next instruction. The relative byte is sign extended so that memory references may be within the range of – 126 and + 129 locations of the instruction. Relative addressing instructions occupy two bytes, the opcode and the relative byte.

$PC + 1 \rightarrow PC$
$(PC) \rightarrow TEMP$
$PC + 1 \rightarrow PC$
$EA = PC + TEMP$ iff branch is taken

Assembly Examples:

```
          0487        A GOOSE   EQU   *
0487 27 04       048D           BEQ   SWAN
0489 20 FC       0487           BRA   GOOSE
048B 20 FE       048B           BRA   *
          048D        A SWAN    EQU   *
04A7                            ORG   $4A7
          04C0        A PROG2   EQU   $4C0
04A7 27 17       04C0           BEQ   PROG2   (See example description below.)
```

Figure 4-7 shows an example of the Relative Addressing Mode. In this example, the program contains an instruction to branch to PROG2, if the condition code register Z-bit was set by a previous program step.

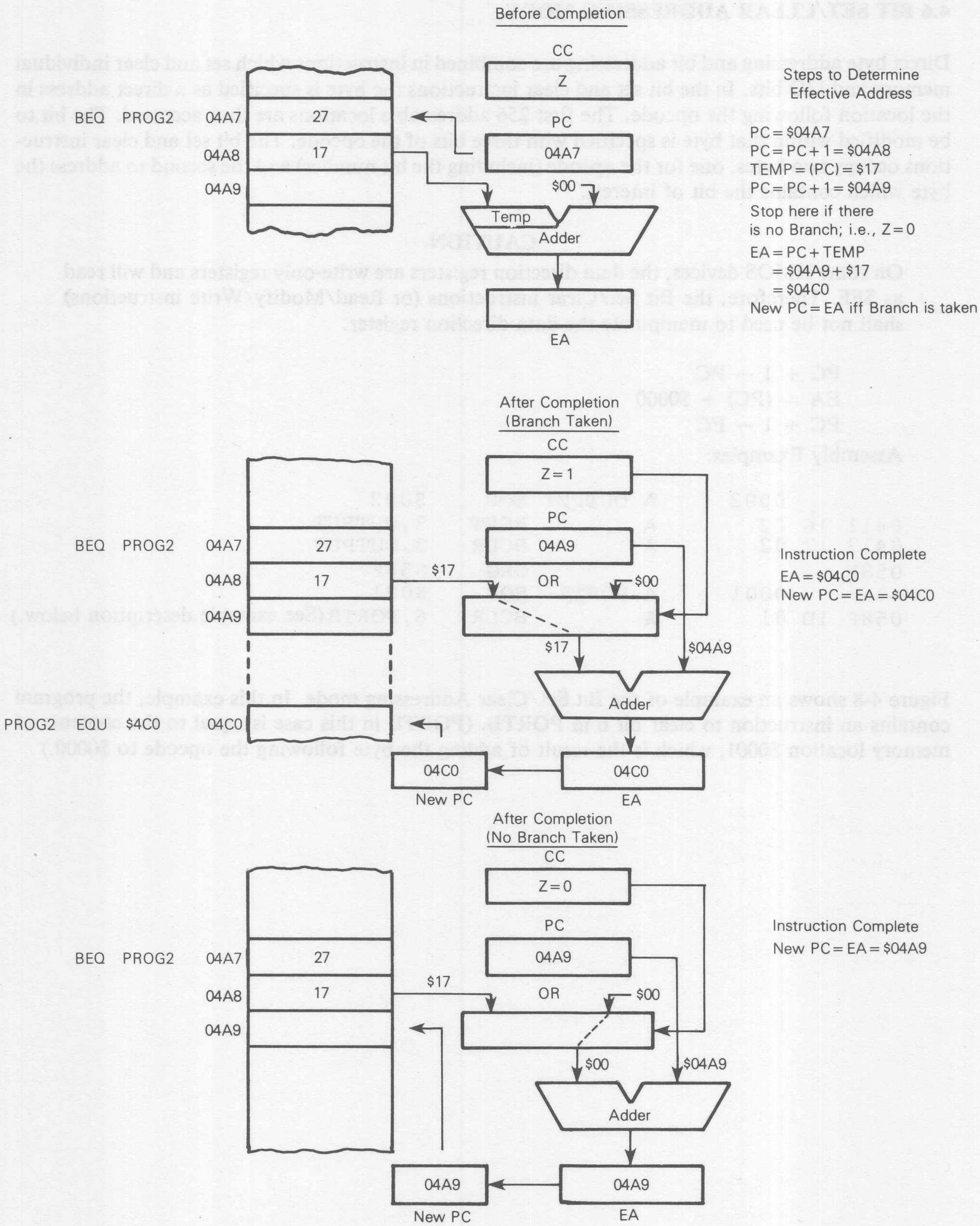


Figure 4-7. Relative Addressing Mode Example

4.6 BIT SET/CLEAR ADDRESSING MODE

Direct byte addressing and bit addressing are combined in instructions which set and clear individual memory and I/O bits. In the bit set and clear instructions the byte is specified as a direct address in the location following the opcode. The first 256 addressable locations are thus accessed. The bit to be modified within that byte is specified with three bits of the opcode. The bit set and clear instructions occupy two bytes, one for the opcode (including the bit number) and the second to address the byte which contains the bit of interest.

CAUTION

On some HMOS devices, the data direction registers are write-only registers and will read as $FF. Therefore, the Bit Set/Clear instructions (or Read/Modify/Write instructions) shall not be used to manipulate the data direction register.

PC + 1 → PC
EA = (PC) + $0000
PC + 1 → PC

Assembly Examples:

```
          0002     A  OUTPUT  EQU     $002
0411  16  02       A          BSET    3,OUTPUT
0413  17  02       A          BCLR    3,OUTPUT
058F               A          ORG     $58F
          0001     A  PORTB   EQU     $001
058F  1D  01       A          BCLR    6,PORTB (See example description below.)
```

Figure 4-8 shows an example of the Bit Set/Clear Addressing mode. In this example, the program contains an instruction to clear bit 6 in PORTB. (PORTB in this case is equal to the contents of memory location $0001, which is the result of adding the byte following the opcode to $0000.)

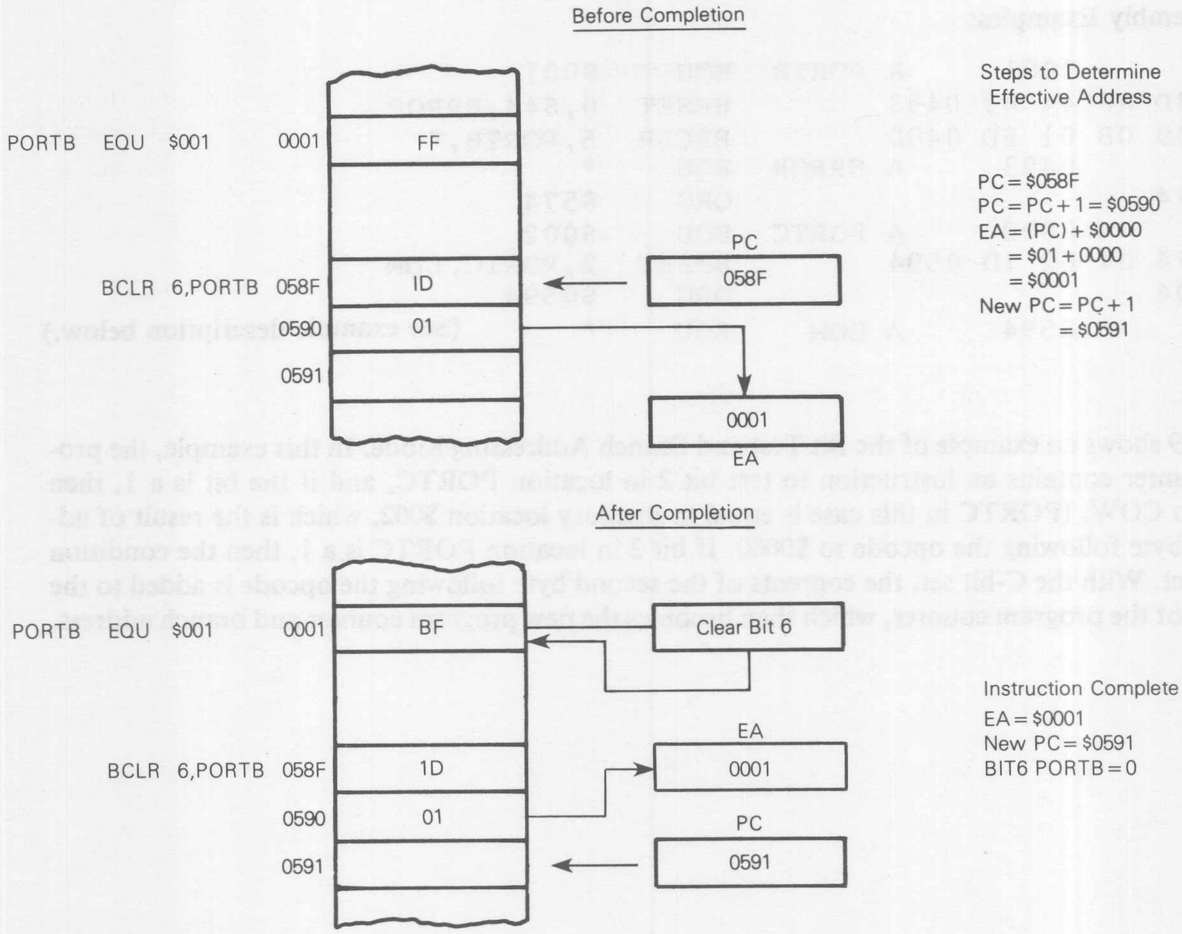


Figure 4-8. Bit Set/Clear Addressing Mode Example

4.7 BIT TEST AND BRANCH ADDRESSING MODE

This mode is a combination of direct, relative and bit addressing. The data byte to be tested is located via a direct address in the location following the opcode. The bit to be tested within the byte is identified within the opcode. The relative address is in the byte following the direct address. The bit test and branch instructions thus occupy three bytes.

NOTE

On some HMOS devices, the data direction registers are write-only registers and will provide an error read of $FF.

$PC + 1 \rightarrow PC$
$EA = (PC) + \$0000$
$PC + 1 \rightarrow PC$
$(PC) \rightarrow TEMP$
$PC + 1 \rightarrow PC$
$EA2 = PC + TEMP$ iff branch is taken

Assembly Examples:

```
          0001        A PORTB    EQU      $001
048D 00  44 03  0493              BRSET    0,$44,ERROR
0490  0B  01  FD  0490            BRCLR    5,PORTB,*
          0493        A ERROR    EQU      *
0574                              ORG      $574
          0002        A PORTC    EQU      $002
0574  04  02  1D  0594            BRSET    2,PORTC,COW
0594                              ORG      $0594
          0594        A COW      EQU      *
```

(See example description below.)

Figure 4-9 shows an example of the Bit Test and Branch Addressing Mode. In this example, the program counter contains an instruction to test bit 2 in location PORTC, and if the bit is a 1, then branch to COW. (PORTC in this case is equal to memory location $002, which is the result of adding the byte following the opcode to $0000. If bit 2 in location PORTC is a 1, then the condition C-bit is set. With the C-bit set, the contents of the second byte following the opcode is added to the contents of the program counter, which then becomes the new program counter and branch address, $0594.)

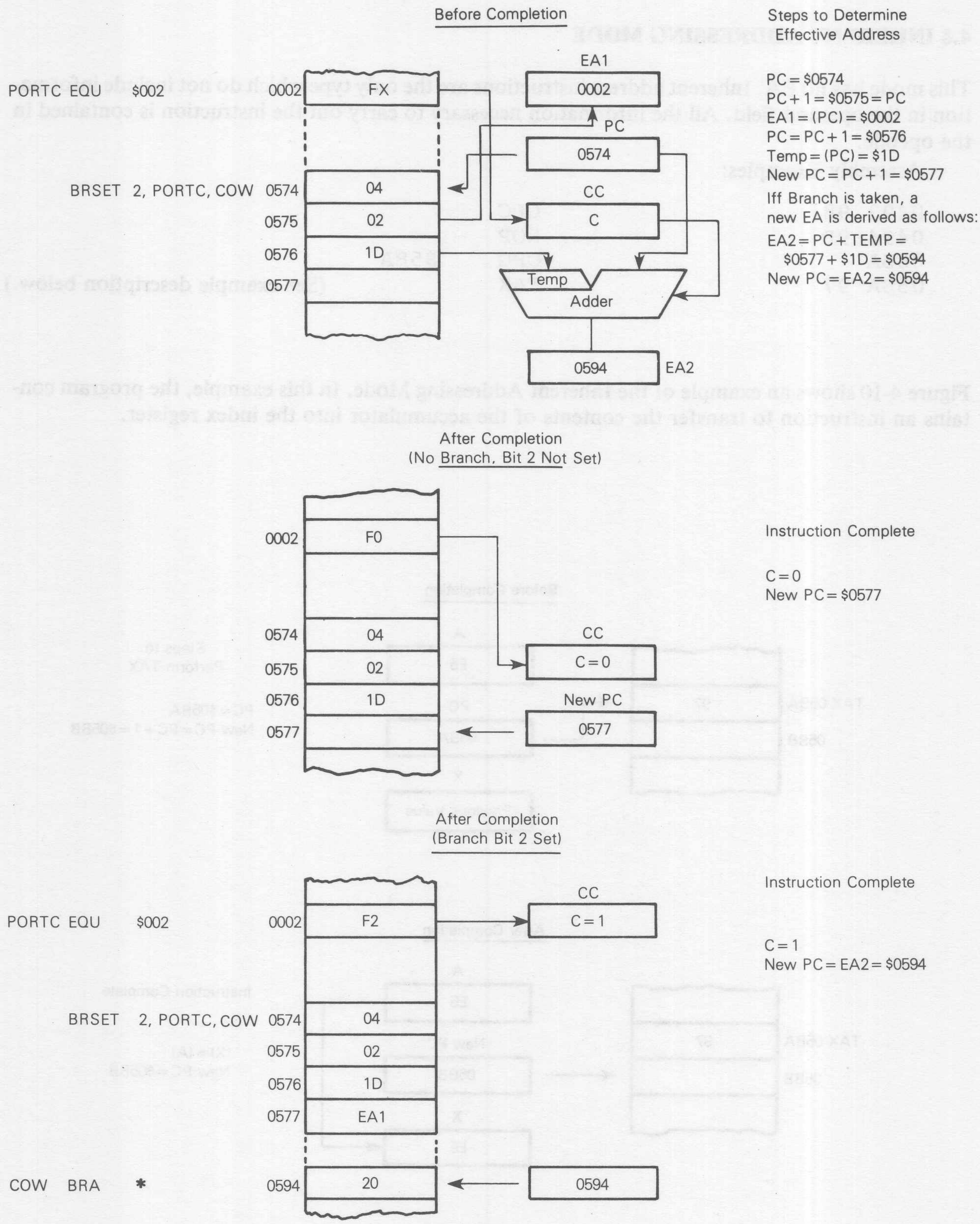


Figure 4-9. Bit Test and Branch Addressing Mode Example

4.8 INHERENT ADDRESSING MODE

This mode has no EA. Inherent address instructions are the only type which do not include information in the operand field. All the information necessary to carry out the instruction is contained in the opcode.

Assembly Examples:

```
0493 98         CLC
0494 9D         NOP
05BA            ORG     $5BA
05BA 97         TAX
```

(See example description below.)

Figure 4-10 shows an example of the Inherent Addressing Mode. In this example, the program contains an instruction to transfer the contents of the accumulator into the index register.

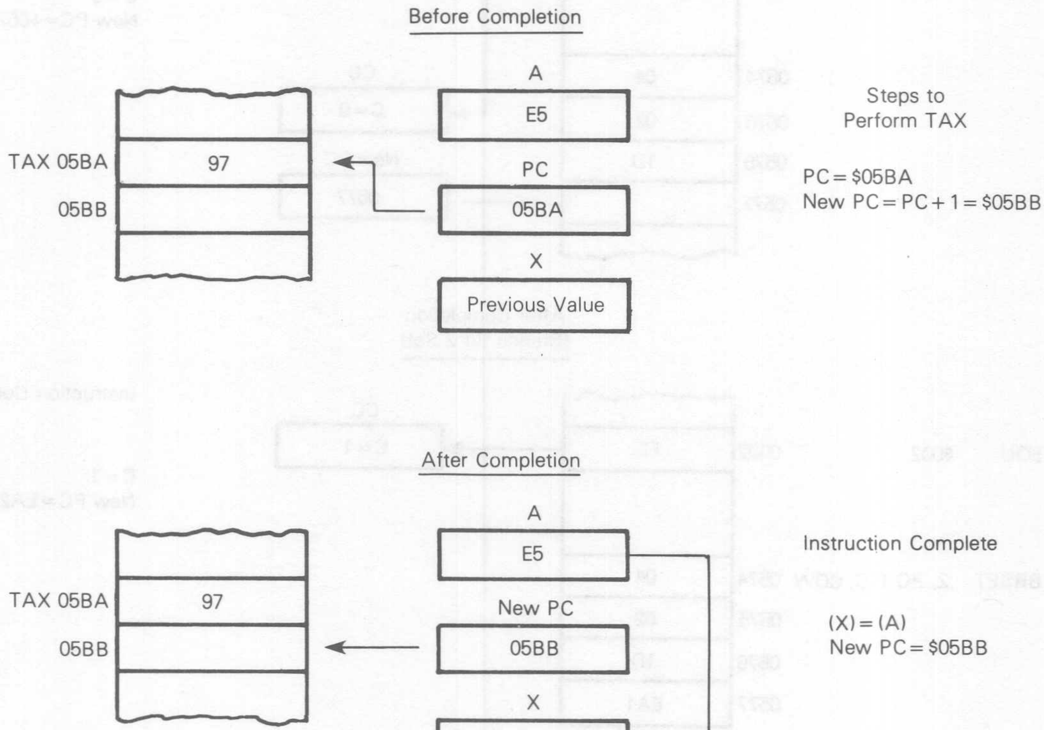


Figure 4-10. Inherent Addressing Mode Example

CHAPTER 5
INSTRUCTION TYPES

5.0 INTRODUCTION

It is convenient to view the M6805 Family as having several instruction types, which are described below. Appendix D contains a detailed definition of the Instruction Set used with the M6805 Family.

5.1 REGISTER/MEMORY INSTRUCTIONS

Most of these instructions contain two operands. One operand is inherently defined as either the accumulator or the index register. The other operand is fetched from memory via one of the addressing modes. The addressing modes which are applicable to the register/memory instructions are given below:

- Immediate
- Direct
- Extended
- Indexed — no offset
- Indexed — 1 byte offset
- Indexed — 2 byte offset

Immediate addressing is not usable with the store and jump instructions (STA, JMP, JSR and STX). A listing of the Register/Memory instructions is given below.

ADC	Add Memory and Carry to Accumulator
ADD	Add Memory to Accumulator
AND	AND Memory with Accumulator
BIT	Bit Test Memory with Accumulator (Logical Compare)
CMP	Compare Accumulator with Memory (Arithmetic Compare)
CPX	Compare Index Register with Memory (Arithmetic Compare)
EOR	Exclusive Or Memory with Accumulator
JMP	Jump
JSR	Jump to Subroutine
LDA	Load Accumulator from Memory
LDX	Load Index Register from Memory
ORA	OR Memory with Accumulator
SBC	Subtract Memory and Borrow from Accumulator

STA	Store Accumulator in Memory
STX	Store Index Register in Memory
SUB	Subtract Memory from Accumulator

5.2 READ/MODIFY/WRITE INSTRUCTIONS

These instructions read a memory location or register, modify or test the contents, and then write the modified value back into the memory or the register. The available addressing modes for these instructions are given below.

Direct
Inherent
Indexed — No Offset
Indexed — 1 byte offset

The Read/Modify/Write instructions are listed below.

ASL	Arithmetic Shift Left (Same as LSL)
ASR	Arithmetic Shift Right
CLR	Clear
COM	Complement
DEC	Decrement
INC	Increment
LSL	Logical Shift Left (Same as ASL)
LSR	Logical Shift Right
NEG	Negate (Two's Complement)
ROL	Rotate Left thru Carry
ROR	Rotate Right thru Carry
TST	Test for Negative or Zero

5.3 BRANCH INSTRUCTIONS

In this set of instructions the program branches to a different routine when a particular condition is met. When the specified condition is not met, execution continues with the next instruction. Most of the branch instructions test the state of one or more of the condition code bits.Relative is the only legal addressing mode applicable to the branch instructions. A list of the branch instructions is provided below.

BCC	Branch iff Carry Clear (Same as BHS)
BCS	Branch iff Carry is Set (Same as BLO)
BEQ	Branch iff Equal to Zero
BHCC	Branch iff Half Carry is Clear
BHCS	Branch iff Half Carry is Set

BHI	Branch iff Higher than Zero
BHS	Branch iff Higher or Same as Zero (Same as BCC)
BIH	Branch iff Interrupt Line is High
BIL	Branch iff Interrupt Line is Low
BLO	Branch iff Lower than Zero (Same as BCS)
BLS	Branch iff Lower or Same as Zero
BMC	Branch iff Interrupt Mask is Clear
BMI	Branch iff Minus
BMS	Branch iff Interrupt Mask is Set
BNE	Branch iff Not Equal to Zero
BPL	Branch iff Plus
BRA	Branch Always
BRN	Branch Never
BSR	Branch to Subroutine

Note that the BIH and BIL instructions permit an external pin to be tested easily.

5.4 BIT MANIPULATION INSTRUCTIONS

There are two basic types of bit manipulation instructions. One group either sets or clears any single bit in a memory byte. This instruction group uses the Bit Set/Clear addressing mode which is similar to direct addressing. The bit number (0-7) is part of the opcode. The other group tests the state of any single bit in a memory location and branches if the bit is set or clear. These instructions have 'test and branch' addressing. The bit manipulation instructions are shown below.

BCLR n	Clear Bit n in Memory
BRCLR n	Branch iff Bit n in Memory is Clear
BRSET n	Branch iff Bit n in Memory is Set
BSET n	Set Bit n in Memory (n = 0...7)

5.5 CONTROL INSTRUCTIONS

Instructions in this group have inherent addressing. These instructions manipulate condition code bits, control stack and interrupt operations, transfer data between the accumulator and index register, and do nothing (NOP). The control instructions are listed below.

CLC	Clear Carry Bit
CLI	Clear Interrupt Mask Bit
NOP	No-Operation
RSP	Reset Stack Pointer
RTI	Return from Interrupt

RTS	Return from Subroutine
SEC	Set Carry Bit
SEI	Set Interrupt Mask Bit
SWI	Software Interrupt
TAX	Transfer Accumulator to Index Register
TXA	Transfer Index Register to Accumulator

CHAPTER 6
PROGRAMMING INTERRUPTS

6.0 INTRODUCTION

One of the major features of the M6805 Family is that it has both hardware and software interrupts. Typically, the hardware interrupts are represented by both external and internal interrupts. The software interrupt (SWI) instruction provides a program initiated interrupt capability.

When an interrupt occurs (either hardware or software), the normal processing is suspended and an interrupt routine is executed. Interrupts, as they occur in the M6805 Family, eliminate the need for inefficient main program "branch on status" loops for both timed and external events.

Since the hardware interrupts are maskable, their effects on the CPU is controllable. All interrupts are latched so that interrupt events are not lost while they are masked. Such interrupt requests are held pending until the mask(s) is cleared. The I-bit status, in the condition code register (CC), controls the masking of all hardware interrupts. Other masks, such as bit 6 in the timer control register, provide additional levels of interrupt masking. Upon completion of the instruction being executed, the hardware controlled sequence will cause the following to be stored (using the stack pointer):

(1) The lower program counter (PCL) value (eight bits)

(2) the upper program counter (PCH) value (up to eight bits)

(3) the index register

(4) the accumulator

(5) the condition code register

Following this register 'push' sequence, the I-bit in the CC register is set which masks further interrupts. In addition, the vector address, stored at a location unique to the interrupt being serviced, is loaded into the PCH and PCL, respectively. This becomes the starting address of the interrupt software service routine. At the end of the interrupt software service routine, a return from interrupt (RTI) instruction is executed. The RTI execution is a 'pull' sequence that restores the state of the CPU. The five bytes saved prior to the interrupt routine are loaded back into the program register. When RTI is complete, the restored PC permits the interrupted program to continue.

6.1 TIMER INTERRUPT

When the timer mask bit in the timer control word is zero, a timer interrupt is generated each time the counter reaches zero, provided the interrupt mask bit in the condition code register is also zero. When the interrupt is recognized, the current machine state is pushed onto the stack and the PC is loaded with the timer interrupt vector address (two bytes). The I-bit in the condition code register is also set, which masks further interrupts. At the end of the execution of the timer interrupt routine, an RTI instruction is executed to restore the machine state and return execution to the interrupted program, with all registers unchanged.

NOTE

One of the tasks, which should be accomplished by the timer interrupt software routine, is to clear the timer interrupt request flag. This flag is stored at $09 (Timer Control Register), bit 7.

6.2 EXTERNAL INTERRUPT

If the I-bit in the condition code register is cleared (interrupts enabled) the external interrupt pin(s) initiate the interrupt sequence. The various M6805 Family MPUs recognize different external interrupt signals. Some of the options are high-to-low transition, a zero-crossing, and a low level. Recognition of the INT external interrupt is much the same as the timer interrupt, except that the vector address is stored in a different memory location.

NOTE

Typically, externally generated interrupt requests are cleared by hardware while the specific interrupt is being serviced. However, certain specific versions of the M6805 Family may contain additional external interrupts. Instructions for clearing these additional external requests are found in the specific Data Sheet instructions (either automatically by hardware or by software control such as the timer interrupt request flag).

6.3 SOFTWARE INTERRUPT

The software interrupt is an executable instruction that behaves much like a hardware interrupt. When the SWI is executed the machine state is saved on the stack and the software interrupt vector is fetched from memory. An SWI will be executed regardless of the state of the I-bit in the condition code register. Software interrupts are used as breakpoints for debugging in many systems.

6.4 RESET

Reset is not an interrupt but behaves much like one. When the reset occurs, the vector, stored in memory, is loaded into the program counter (PC). During reset, the I-bit in the condition code register and the timer interrupt mask bit in (in the TCR) are both set. Also, the stack pointer is reset to the beginning of the stack. In addition, the timer and its prescaler are set to all 1's and the Data Direction Registers are cleared on all I/O ports (outputs assume high impedance state). The contents of the reset vector location contains the address of the first instruction to be executed after reset.

6.5 VECTORS

A vector is the address from which the next instruction will be fetched. To summarize, the basic vectors for the M6805 Family are:

$XXF8	Timer
$XXFA	INT
$XXFC	SWI
$XXFE	Reset

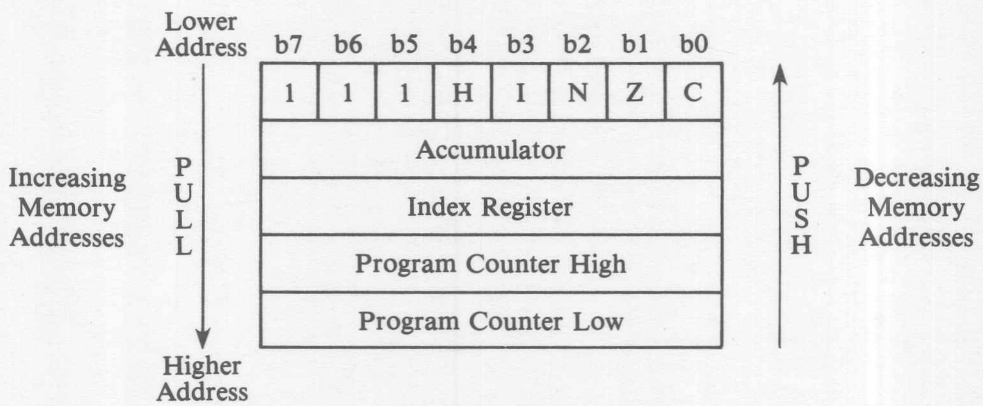
NOTE

XX refers to the top of the available memory address space and varies by family part — consult specific device data sheet. For example: in the 'PZ device (11-bit PC) XX = 07; in the 'RZ device (12-bit PC) XX = 0F; and in the 'EZ device (13-bit PC) XX2 = 1F.

The total number of basic vectors may increase depending upon the specific family I/O options (refer to specific device data sheet).

6.6 STACKING ORDER

The machine state is pushed onto or pulled from the stack in the following order:



Since the stack pointer decrements during pushes the PCL is stacked first, then the PCH, etc. Pulling from the stack is in the reverse order. The stack pointer in the M6805 Family always points to the next free location on the stack (similar to the MC6800 and MC6801).

APPENDIX A
M6800 COMPATIBILITY

A.0 INTRODUCTION

Strictly speaking, the M6805 Family is neither source nor object code compatible with the MC6800; but it is very similar to all M6800 family processors. An experienced MC6800 programmer should have little difficulty adapting to the M6805 Family instruction set. The following paragraphs enumerate the differences between the MC6800 and the MC6805.

A.1 REMOVED B-REGISTER

In order to free up valuable opcode space, the B-register is removed in the MC6805. Therefore, none of the register/memory or read/modify/write instructions have a B-register form. Several other instructions are also not available in the MC6805, including:

SBA, CBA, TAB, TBA, ABA, PSHB, and PULB

A.2 REMOVED V-FLAG

The V-flag bit and the logic to set it is removed in the MC6805. This was done because usage of the small controller does not generally require signed arithmetic operations. However, unsigned arithmetic operations are still available. Without the V-flag bit, the following MC6800 instructions are not available in the MC6805.

SEV, CLV, BVC, BVS, BGE, BLT, BGT, and BLE.

Notice that the unsigned inequalities are still available using BHS (BCC) and BLO (BCS).

A.3 REDUCED STACK CONTROL

Instructions relating to the manipulation of the SP are greatly reduced. On reset, or upon execution of the RSP instruction, the SP is initialized to $7F. Other instructions that were deleted include:

LDS, STS, INS, DES, PSHA, PULA, TXS, TSX and WAI.

A.4 REMOVED DAA

Although the DAA is useful in some low-end applications, it was deleted. The H-bit, however, was retained and two additional branches were added to branch if the H-bit is set or cleared (BHCS, BHCC). These branches can be used to write software subroutines accomplishing DAA (remember, ROM is much cheaper than the DAA).

A.5 CHANGED REGISTER LENGTHS

The X-register was reduced to eight bits, the SP to eight bits or less and the PC to 16 bits or less. The change in the X-register size from 16 to eight bits required changes in the addressing modes; these are described in the Addressing Modes Chapter. Also, since the X and A registers are equal in size, two new instructions are added to transfer X to A and A to X (TXA, TAX).

A.6 BIT MANIPULATION

Bit manipulation instructions are added to the MC6805 because they are extremely useful for low-end applications. Two classes of bit manipulation instructions were added, Bit Set/Clear, and Test and Branch on Bit Set/Clear.

(a) Bit Set/Clear

These instructions allow any bit in page zero (< $100) including bits in the I/O ports (but not always the data direction registers) to be set or cleared with one 2-byte instruction.

(b) Test and Branch on Bit Set/Clear

These instructions test any bit in page zero, including I/O, RAM and ROM, and branch, if the bit is set or cleared. In addition, the C-bit of the Condition Code Register contains the state of the bit tested.

A.7 NEW BRANCHES

Several new branches are added to facilitate low-end type programs. BHCS and BHCC are useful in BCD additions. A branch, if interrupt mask bit is set or cleared (BMS/BMC), is also added. This eliminates the need for TAP and TPA since each bit in the condition code register can be tested by a branch. Two more branches are added that branch on the logic condition of the interrupt line (high or low): BIH/BIL. These allow the interrupt line to be used as an additional input in systems not using interrupts.

A.8 NEW ADDRESSING MODES

The addressing modes of the MC6800 were optimized for the MC6805. For more details see the Addressing Modes section of the manual.

A.9 READ/MODIFY/WRITE THE X-REGISTER

By utilizing the column in the opcode map vacated by the B-register for read/modify/write, and since the X-register is now eight bits, all of these operations are available to the X-register. For example:

ROLX, INCX, CLRX, NEGX, etc.

This eliminated the traditional INX, DEX. However, mnemonics INX and DEX are still recognized by the assembler for compatibility.

A.10 CONVENIENCE MNEMONICS

These are not new M6805 Family instructions, but only represent improvements to the MC6805 assembler that allow existing instructions to be recognized by more than mnemonic.

(a) LSL (Logical Shift Left)

Since logical and arithmetic left shifts are identical, LSL is equivalent to ASL.

(b) BHS (Branch Higher or Same)

After a compare or subtract, the carry is cleared if the register argument was higher or equal to the memory argument, hence the BHS is equivalent to BCC.

(c) BLO (Branch if Lower)

After a compare or subtract, the carry is set if the register argument was lower than the memory argument, hence the BLO is equivalent to BCS.

APPENDIX B
HMOS AND CMOS TECHNOLOGIES

B.1 HMOS LOGIC

The HMOS inverter circuit, shown in Figure B-1, illustrates the operating principles of HMOS logic. Two transistors are series connected between ground (V_{SS}) and V_{DD}; one is an active N-channel transistor and the other is a turned-on pull-up transistor. When a logic low is applied to the circuit input, the N-channel transistor is reverse biased and represents a high impedance, compared to the pull-up transistor (which provides the same function as a resistor). A load connected to the circuit output can be driven to a logic high by the supply current through the pull-up transistor.

When a logic high is applied to the circuit input, the N-channel transistor is turned on and becomes a very low resistance to V_{SS} causing the output to go low and a current, equal to $V_{DD} - V_{SS}/R$, to flow through the pull-up transistor.

Other logic circuits constructed in HMOS technology use series and parallel combinations of the N-channel transistors. However, they all rely on the same operating principle, that is, the active N-channel transistor is used to sink current from the output and a passive load transistor, which behaves similarly to a resistor, is used to source current to the output.

It is the current flowing through the pull-up load transistor, when the N-channel transistor is turned on, that accounts for most of the power consumed in an HMOS integrated circuit.

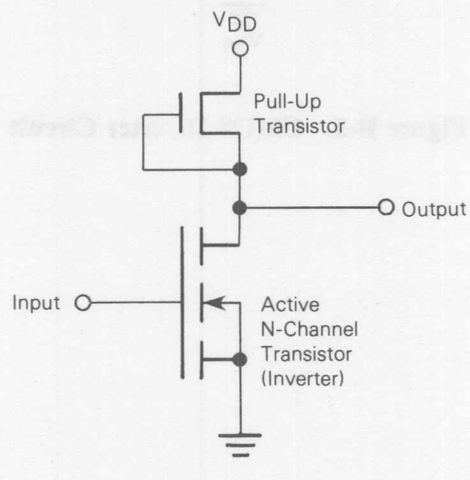


Figure B-1. HMOS Inverter Circuit

B.2 CMOS LOGIC

The CMOS inverter circuit, shown in Figure B-2, illustrates the operating principles of CMOS logic. In CMOS, the pull-up transistor is replaced with an active, P-channel, transistor. In this type of circuit, one transistor complements the other; i.e., when one is turned on the other is turned off. The characteristics of the P-channel transistor are such that a high signal input turns it off; conversely, a low signal input turns it on.

The active P-channel transistor sources current when the output is high (input low), and presents a high impedance when the output is low (input high). Thus, there is essentially no current flow within the inverter whenever the output is low. The overall result is extremely low power consumption because there is no power loss through the active pull-up transistor.

The switch point of the CMOS inverter is at approximately 50% of the supply voltage (V_{DD}) rather than being determined by the threshold of the N-channel transistor. Because of this, the operating voltage range of a CMOS device is much wider than that of an HMOS device. This permits a greater choice of supply voltages or allows the use of a less regulated power supply.

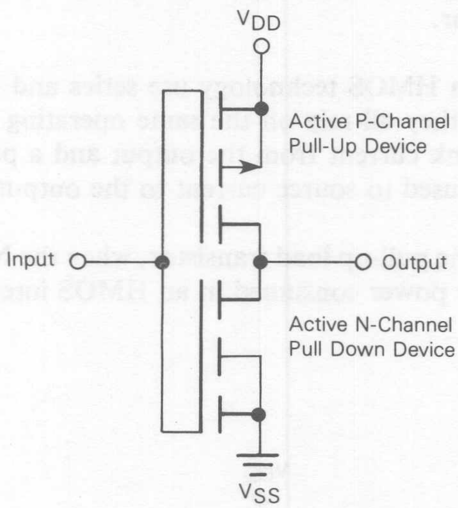


Figure B-2. CMOS Inverter Circuit

APPENDIX C
RASM05 MACRO ASSEMBLER
SYNTAX AND DIRECTIVES

C.0 ASSEMBLY LANGUAGE SYNTAX AND ASSEMBLER DIRECTIVES

This appendix provides information concerning the Assembly Language Syntax and Assembler Directive for the M6805 Family. This information is more thoroughly discussed in *Macro Assemblers Reference Manual* M68MASR(D2) for M6800, 6801, 6805 and 6809; Motorola Literature Distribution Center, Phoenix, Az.

M6805 Family assembly language source statements follow the same format as M6800 source statements. See *Macro Assembler Reference Manual* M68MASR(D2) for detailed M6805 Family syntax. Highlights of the M6805 Family syntax and assembler directives are discussed in the following paragraphs.

C.1 OPERATION FIELD SYNTAX

All instruction mnemonics for the M6805 Family are three, four, or five characters long. Examples are:

LDA
JSR
INC
BHCC
BRSET

If the accumulator or index register is used as the operand of read/modify/write instructions, then the register is appended to the operation field. For example:

NEGA
RORX
INCX
DECA
TSTA

C.2 OPERAND FIELD SYNTAX

C.2.1 Inherent

Inherent instructions are the only type which do not include information in the operand field. All information necessary is incorporated in the operation field. Some examples are listed below. Note that an "A" or an "X" is added to the opcode for the register reference inherent instructions.

```
RTS
CLC
INCA
RORA
INCX
RORX
```

C.2.2 Immediate

The immediate value appears in the operand field preceded by a '#'. Example:

```
LDA     #30
LDX     #$49
CPX     #$FF
LDA     #ADDR
```

C.2.3 Direct Addressing

The direct address appears in the operand field. If, on any pass through the source program, the assembler finds an unresolved (undefined) forward reference, the longer extended addressing mode is chosen instead of the direct addressing mode even if the address is subsequently found to be on page zero. To ensure direct addressing for direct variables, always define the variable before using it. In read/modify/write instructions all addresses are assumed to be direct since extended addressing is illegal with this mode. Examples:

```
LDA     CAT
STA     $30
CPX     DOG
ROL     $01
```

Where CAT and DOG have addresses <$100.

C.2.4 Extended Addressing

The extended address appears in the operand field. This mode is only legal when executing register/memory instructions. Examples:

```
LDA      BIG
LDA      $325
STA      COW
```

Where BIG and COW have addresses >$100.

C.2.5 Indexed — No Offset

The characters comma and X appear in the operand field. For example:

```
LDA      ,X
COM      ,X
STA      ,X
INC      ,X
TST      ,X
```

C.2.6 Indexed — One Byte Offset

The offset appears followed by a comma and "X". The offset must have a value <$100. Examples:

```
LDA      3, X
LDA      TABLE, X
INC      50, X
```

Where TABLE <$100.

C.2.7 Indexed — Two Byte Offset

The offset appears followed by a comma and "X". The offset would normally have a value >$100. Examples:

```
LDA      300, X
LDA      ZOT, X
COM      500, X
```

Where ZOT >$100.

C.2.8 Bit Set/Clear

The bit set and clear instructions contain the bit number followed by a comma and the address. Examples:

BSET	3, CAT
BCLR	4, $30
BCLR	5, DOG

Where CAT and DOG are <$100.

C.2.9 Bit Test and Branch

The bit test and branch instructions contain the bit number, a comma, the address to be tested, a comma, and the location to branch to if the test was successful. Examples:

PIG	BRSET	3, CAT, DOG
DOG	BRCLR	4, CAT, PIG

Where CAT <$100, DOG and PIG are Relative Addresses similar to those explained in the next paragraph.

C.2.10 Relative Addressing

The operand field contains the label of the address to be loaded into the program counter if the branch is taken. The branch address must be in the range −126 to +129. Examples:

BEQ	CAT
BNE	DOG
BRA	PIG

C.3 ASSEMBLER DIRECTIVE SUMMARY

The assembler directives are instructions to the assembler rather than instructions which are directly translated into object code. Detailed descriptions are provided in the M68MASR(D2) reference manual.

C.3.1 Assembly Control Directives

END	Program end
FAIL	Programmer generated errors
NAM	Assign program name
ORG	Origin program counter

C.3.2 Symbol Definition Directives

ENDM	Macro definition end
EQU	Assign permanent value
MACR	Macro definition start
SET	Assign temporary value

C.3.3 Data Definition/Storage Allocation Directives

BSZ	Block storage of zero; single bytes
FCB	Form constant byte
FCC	Form constant character string
FDB	Form constant double byte
RMB	Reserve memory; single bytes

C.3.4 Program Relocation Directives

ASCT	Absolute section
BSCT	Base section
COMM	Named common section
CSCT	Blank common section
DSCT	Data section
IDNT	Identification record
PSCT	Program section
OPT REL	Relocatable output selected
XDEF	External symbol definition
XREF	External symbol reference

C.3.5 Conditional Assembly Directives

ENDC	End of current level of conditional assembly
IFC	Assemble if strings compare
IFEQ	Assemble if expression is equal to zero
IFGE	Assemble if expression is greater than or equal to zero
IFGT	Assemble if expression is greater than zero
IFLE	Assemble if expression is less than or equal to zero
IFLT	Assemble if expression is less than zero
IFNC	Assemble if strings do not compare
IFNE	Assemble if expression is not equal to zero

C.3.6 Listing Control Directives

OPT ABS	Select absolute MDOS-loadable object output
OPT CL	Print conditional assembly directives
OPT NOCL	Don't print conditional assembly directives
OPT CMO	Allow CMOS instructions STOP and WAIT (M6805 only)
OPT NOCMO	Don't allow CMOS instructions STOP and WAIT (M6805 only)
OPT CRE	Print cross reference table
OPT G	Print generated lines of FCB, FCC, and FDB directives
OPT NOG	Don't print generated lines of FDB, FCC, and FDB directives
OPT L	Print source listing from this point
OPT NOL	Inhibit printing of source listing from this point
OPT LLE=n	Change line length
OPT LOAD	Select absolute EXORciser-loadable object output
OPT M	Create object output in memory
OPT MC	Print macro calls
OPT NOMC	Don't print macro calls
OPT MD	Print macro definitions
OPT NOMD	Don't print macro definitions
OPT MEX	Print macro expansions
OPT NOMEX	Don't print macro expansions
OPT O	Create object output file
OPT NOO	Do not create object output file
OPT P=n	Change page length
OPT NOP	Inhibit paging and printing of headings
OPT REL	Select relocatable object output
OPT S	Print symbol table
OPT SE	Print user-supplied sequence numbers
OPT U	Print unassembled code from conditional directives
OPT NOU	Don't print unassembled code from conditional directives
PAGE	Print subsequent statements on top of next page
SPC	Skip lines
TTL	Initialize heading for source listing

APPENDIX D
INSTRUCTION SET
DETAILED DEFINITION

D.0 EXECUTABLE INSTRUCTIONS

D.1 INTRODUCTION

In the pages that follow this section, the various Accumulator and Memory operations, together with the respective Mnemonic, provides a heading for each of the executable instructions. The STOP and WAIT instructions apply only to the CMOS M146805 Family. The pages are arranged in alphabetical order of the Mnemonic. A brief description of the operation is provided along with other applicable pertinent information, including: condition code status; Boolean Formula; Source Forms; usable Addressing Modes; number of execution cycles (both M6805 and M146805 Families), number of bytes required; and the opcode for each usable Addressing Mode. Paragraph D.2 contains a listing of the various nomeclature (abbreviations and signs) used in the operations.

D.2 NOMENCLATURE

The following nomenclature is used in the executable instructions which follow this paragraph.

(a) Operators:

( )	indirection. i.e., (SP) means the value pointed to by SP
←	is loaded with (read: 'gets')
•	boolean AND
v	boolean (inclusive) OR
⊕	boolean EXCLUSIVE OR
~	boolean NOT
−	negation (two's complement)

(b) Registers in the MPU:

ACCA	Accumulator
CC	Condition Code Register
X	Index Register
PC	Program Counter
PCH	Program Counter High Byte
PCL	Program Counter Low Byte
SP	Stack Pointer

(c) Memory and Addressing:

M	Contents of any memory location (one byte)
Rel	Relative address (i.e., the two's complement number stored in the second byte of machine code in a branch instruction.)

(d) Bits in the Condition Code Register:

C	Carry/Borrow, Bit 0
Z	Zero Indicator, Bit 1
N	Negative Indicator, Bit 2
I	Interrupt Mask, Bit 3
H	Half Carry Indicator, Bit 4

(e) Status of Individual Bits BEFORE Execution of an Instruction

An	Bit n of ACCA (n = 7, 6, 5, 4, 3, 2, 1, 0)
Xn	Bit n of X (n = 7, 6, 5, 4, 3, 2, 1, 0)
Mn	Bit n of M (n = 7, 6, 5, 4, 3, 2, 1, 0). In read/modify/write instructions, Mn is used to represent bit n of M, A or X.

(f) Status of Individual Bits AFTER Execution of an Instruction:

Rn	Bit n of the result (n = 7, 6, 5, 4, 3, 2, 1, 0)

(g) Source Forms:

P	Operands with IMMediate, DIRect, EXTended and INDexed (0, 1, 2 byte offset) addressing modes
Q	Operands with DIRect, INDexed (0 and 1 byte offset) addressing modes
dd	Relative operands
DR	Operands with DIRect addressing mode only.

(h) iff

abbreviation for if-and-only-if.

ADC

Add with Carry

Operation: ACCA ← ACCA + M + C

Description: Adds the contents of the C bit to the sum of the contents of ACCA and M, and places the result in ACCA.

Condition Codes:

- H: Set if there was a carry from bit 3; cleared otherwise.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if there was a carry from the most significant bit of the result; cleared otherwise.

Boolean Formulae for Condition Codes:

$H = A3 \bullet M3 v M3 \bullet \overline{R3} v \overline{R3} \bullet A3$

$N = R7$

$Z = \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

$C = A7 \bullet M7 v M7 \bullet \overline{R7} v \overline{R7} \bullet A7$

Source Form(s): ADC P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A9
Direct	4	3	2	B9
Extended	5	4	3	C9
Indexed 0 Offset	4	3	1	F9
Indexed 1-Byte	5	4	2	E9
Indexed 2-Byte	6	5	3	D9

ADD ADD

Add

Operation: ACCA ← ACCA + M

Description: Adds the contents of ACCA and the contents of M and places the result in ACCA.

Condition Codes:
- H: Set if there was a carry from bit 3; cleared otherwise.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if there was a carry from the most significant bit of the result; cleared otherwise.

Boolean Formulae for Condition Codes:

$H = A3 \bullet M3 v M3 \bullet \overline{R3} v \overline{R3} \bullet A3$

$N = R7$

$Z = \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

$C = A7 \bullet M7 v M7 \bullet \overline{R7} v \overline{R7} \bullet A7$

Source Form(s): ADD P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	AB
Direct	4	3	2	BB
Extended	5	4	3	CB
Indexed 0 Offset	4	3	1	FB
Indexed 1-Byte	5	4	2	EB
Indexed 2-Byte	6	5	3	DB

AND Logical AND AND

Operation: ACCA ← ACCA . M

Description: Performs logical AND between the contents of ACCA and the contents of M and places the result in ACCA. Each bit of ACCA after the operation will be the logical AND result of the corresponding bits of M and of ACCA before the operation.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if the most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
C: Not affected.

Boolean Formulae for Condition Codes:

$N = R7$
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): AND P

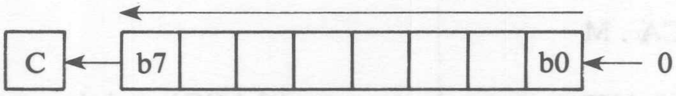
Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A4
Direct	4	3	2	B4
Extended	5	4	3	C4
Indexed 0 Offset	4	3	1	F4
Indexed 1-Byte	5	4	2	E4
Indexed 2-Byte	6	5	3	D4

ASL

Arithmetic Shift Left

ASL

Operation:



Description: Shifts all bits of ACCA, X or M one place to the left. Bit 0 is loaded with a zero. The C bit is loaded from the most significant bit of ACCA, X or M.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if the most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
C: Set if, before the operation, the most significant bit of ACCA, X or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$
$C = M7$

Comments: Same opcode as LSL

Source Form(s): ASL Q, ASLA, ASLX

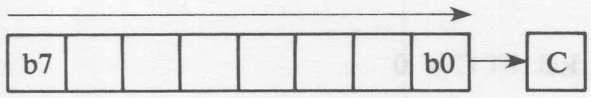
Addressing Mode	Cycles HMOS	CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	48
Index Register	4	3	1	58
Immediate				
Direct	6	5	2	38
Extended				
Indexed 0 Offset	6	5	1	78
Indexed 1-Byte	7	6	2	68
Indexed 2-Byte				

ASR

Arithmetic Shift Right

ASR

Operation:



Description: Shifts all bits of ACCA, X or M one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C bit.

Condition Codes:

H:	Not affected.
I:	Not affected.
N:	Set if the most significant bit of the result is set; cleared otherwise.
Z:	Set if all bits of the result are cleared; cleared otherwise.
C:	Set if, before the operation, the least significant bit of ACCA, X or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

$C = M0$

Source Form(s): ASR Q, ASRA, ASRX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	47
Index Register	4	3	1	57
Immediate				
Direct	6	5	2	37
Extended				
Indexed 0 Offset	6	5	1	77
Indexed 1-Byte	7	6	2	67
Indexed 2-Byte				

BCC **Branch if Carry Clear** BCC

Operation: PC ← PC + 0002 + Rel iff C = 0

Description: Tests the state of the C bit and causes a branch iff C is clear. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: Same opcode as BHS

Source Form(s): BCC dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	24
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BCLR n — Bit Clear Bit n — BCLR n

Operation: Mn ← 0

Description: Clear bit n (n = 0, 7) in location M. All other bits in M are unaffected.

Condition Codes: Not affected.

Source Form(s): BCLR n, DR

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate				
Direct	7	5	2	11+2•n
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BCS Branch if Carry Set BCS

Operation: PC ← PC + 0002 + Rel iff C = 1

Description: Tests the state of the C bit and causes a branch iff C is set. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: Same opcode as BLO

Source Form(s): BCS dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	25
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BEQ — Branch if Equal — BEQ

Operation: PC ← PC + 0002 + Rel iff Z = 1

Description: Tests the state of the Z bit and causes a branch iff Z is set. Following a compare or subtract instruction BEQ will cause a branch if the arguments were equal. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BEQ dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	27
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BHCC — Branch if Half Carry Clear — BHCC

Operation: PC ← PC + 0002 + Rel iff H = 0

Description: Tests the state of the H bit and causes a branch iff H is clear. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BHCC dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	28
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BHCS

Branch if Half Carry Set

BHCS

Operation: PC ← PC + 0002 + Rel iff H = 1

Description: Tests the state of the H bit and causes a branch iff H is set. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BHCS dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	29
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BHI Branch if Higher BHI

Operation: PC ← PC + 0002 + Rel iff (C v Z) = 0
i.e., if ACCA > M (unsigned binary numbers)

Description: Causes a branch iff both C and Z are zero. If the BHI instruction is executed immediately after execution of either of the CMP or SUB instructions, the branch will occur if and only if the unsigned binary number represented by the minuend (i.e., ACCA) was greater than the unsigned binary number represented by the subtrahend (i.e., M). See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BHI dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	22
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BHS — Branch iff Higher or Same — BHS

Operation: PC ← PC + 0002 + Rel iff C = 0

Description: Following an unsigned compare or subtract, BHS will cause a branch iff the register was higher than or the same as the location in memory. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: Same opcode as BCC

Source Form(s): BHS dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	24
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BIH — Branch iff Interrupt Line is High — BIH

Operation: PC ← PC + 0002 + Rel iff INT = 1

Description: Tests the state of the external interrupt pin and branches iff it is high. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: In systems not using interrupts, this instruction and BIL can be used to create an extra I/O input bit. This instruction does NOT test the state of the interrupt mask bit nor does it indicate whether an interrupt is pending. All it does is indicate whether the INT line is high.

Source Form(s): BIH dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	2F
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BIL — Branch if Interrupt Line is Low — BIL

Operation: PC ← PC + 0002 + Rel iff INT = 0

Description: Tests the state of the external interrupt pin and branches iff it is low. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: In systems not using interrupts, this instruction and BIH can be used to create an extra I/O input bit. This instruction does NOT test the state of the interrupt mask bit nor does it indicate whether an interrupt is pending. All it does is indicate whether the INT line is Low.

Source Form(s): BIL dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	2E
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BIT — Bit Test Memory with Accumulator — BIT

Operation: ACCA • M

Description: Performs the logical AND comparison of the contents of ACCA and the contents of M and modifies the condition codes accordingly. The contents of ACCA and M are unchanged.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result of the AND is set; cleared otherwise.
- Z: Set if all bits of the result of the AND are cleared; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): BIT P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A5
Direct	4	3	2	B5
Extended	5	4	3	C5
Indexed 0 Offset	4	3	1	F5
Indexed 1-Byte	5	4	2	E5
Indexed 2-Byte	6	5	3	D5

BLO Branch if Lower BLO

Operation: PC ← PC + 0002 + Rel iff C = 1

Description: Following a compare, BLO will branch iff the register was lower than the memory location. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: Same opcode as BCS

Source Form(s): BLO dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	25
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BLS — Branch iff Lower or Same — BLS

Operation: PC ← PC + 0002 + Rel iff (C v Z) = 1
i.e., if ACCA ← M (unsigned binary numbers)

Description: Causes a branch if (C is set) OR (Z is set). If the BLS instruction is executed immediately after execution of either of the instructions CMP or SUB, the branch will occur if and only if the unsigned binary number represented by the minuend (i.e., ACCA) was less than or equal to the unsigned binary number represented by the subtrahend (i.e., M). See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BLS dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	23
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BMC

Branch if Interrupt Mask is Clear

Operation: PC ← PC + 0002 + Rel iff I = 0

Description: Tests the state of the I bit and causes a branch iff I is clear. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: This instruction does NOT branch on the condition of the external interrupt line. The test is performed only on the interrupt mask bit.

Source Form(s): BMC dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	2C
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BMI Branch if Minus BMI

Operation: PC ← PC + 0002 + Rel iff N = 1

Description: Tests the state of the N bit and causes a branch iff N is set. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s) BMI dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	2B
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BMS — Branch if Interrupt Mask Bit is Set — BMS

Operation: PC ← PC + 0002 + Rel iff I = 1

Description: Tests the state of the I bit and causes a branch iff I is set. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Comments: This instruction does NOT branch on the condition of the external interrupt line. The test is performed only on the interrupt mask bit.

Source Form(s): BMS dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	2D
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BNE — Branch if Not Equal — BNE

Operation: PC ← PC + 0002 + Rel iff Z = 0

Description: Tests the state of the Z bit and causes a branch iff Z is clear. Following a compare or subtract instruction BNE will cause a branch if the arguments were different. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BNE dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	26
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BPL — Branch if Plus — BPL

Operation: PC ← PC + 0002 + Rel iff N = 0

Description: Tests the state of the N bit and causes a branch iff N is clear. See BRA instruction for further details of the execution of the branch.

Condition Codes: Not affected.

Source Form(s): BPL dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	2A
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BRA

Branch Always

Operation: PC ← PC + 0002 + Rel

Description: Unconditional branch to the address given by the foregoing formula, in which Rel is the relative address stored as a two's complement number in the second byte of machine code corresponding to the branch instruction.

NOTE: The source program specifies the destination of any branch instruction by its absolute address, either as a numerical value or as a symbol or expression which can be evaluated by the assembler. The assembler obtains the relative address Rel from the absolute address and the current value of the program counter.

Condition Codes: Not affected.

Source Form(s): BRA dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	20
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BRCLR n Branch if Bit n is Clear BRCLR n

Operation: PC ← PC + 0003 + Rel iff bit n of M is zero

Description: Tests bit n (n = 0, 7) of location M and branches iff the bit is clear.

Condition Codes:
H: Not affected.
I: Not affected.
N: Not affected.
Z: Not affected.
C: Set if Mn = 1; cleared otherwise.

Boolean Formulae for Condition Codes:
C = Mn

Comments: The C bit is set to the state of the bit tested. Used with an appropriate rotate instruction, this instruction is an easy way to do serial to parallel conversions.

Source Form(s): BRCLR n, DR, dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	10	5	3	01+2•n
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BRN — Branch Never — BRN

Description: Never branches. Branch never is a 2 byte 4 cycle NOP.

Condition Codes: Not affected.

Comments: BRN is included here to demonstrate the nature of branches on the M6805 Family. Each branch is matched with an inverse that varies only in the least significant bit of the opcode. BRN is the inverse of BRA. This instruction may have some use during program debugging.

Source Form(s): BRN dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	4	3	2	21
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BRSET Branch if Bit n is Set BRSET

Operation: PC ← PC + 0003 + Rel iff Bit n of M is not zero

Description: Tests bit n (n = 0, 7) of location M and branches iff the bit is set.

Condition Codes:

H: Not affected.
I: Not affected.
N: Not affected.
Z: Not affected.
C: Set if Mn = 1; cleared otherwise.

Boolean Formulae for Condition Codes:

C = Mn

Comments: The C bit is set to the state of the bit tested. Used with an appropriate rotate instruction, this instruction is an easy way to provide serial to parallel conversions.

Source Form(s): BRSET n, DR, dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	10	5	3	2•n
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BSET n — Set Bit in Memory — BSET n

Operation: Mn ← 1

Description: Set bit n (n = 0, 7) in location M. All other bits in M are unaffected.

Condition Codes: Not affected.

Source Form(s): BSET n, DR

Addressing Modes	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate				
Direct	7	5	2	10+2•n
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

BSR — Branch to Subroutine — BSR

Operation:

PC ← PC + 0002
(SP) ← PCL; SP ← SP − 0001
(SP) ← PCH; SP ← SP − 0001
PC ← PC + Rel

Description: The program counter is incremented by 2. The least (low) significant byte of the program counter contents is pushed onto the stack. The stack pointer is then decremented (by one). The most (high) signficant byte of the program counter contents is then pushed onto the stack. Unused bits in the Program Counter high byte are stored as 1's on the stack. The stack pointer is again decremented (by one). A branch then occurs to the location specified by the relative offset. See the BRA instruction for details of the branch execution.

Condition Codes: Not affected.

Source Form(s): BSR dd

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative	8	6	2	AD
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

Clear Carry Bit

Operation: C bit ← 0

Description: Clears the carry bit in the processor condition code register.

Condition Codes:

H: Not affected.
I: Not affected.
N: Not affected.
Z: Not affected.
C: Cleared.

Boolean Formulae for Condition Codes:

C = 0

Source Form(s): CLC

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	98
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

CLI — Clear Interrupt Mask Bit — CLI

Operation: I bit ← 0

Description: Clears the interrupt mask bit in the processor condition code register. This enables the microprocessor to service interrupts. Interrupts that were pending while the I bit was set will now begin to have effect.

Condition Codes:

H: Not affected.
I: Cleared
N: Not affected.
Z: Not affected.
C: Not affected.

Boolean Formulae for Condition Codes:

I = 0

Source Form(s): CLI

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	9A
Relative				
Accumulator				
Index Registers				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

CLR Clear CLR

Operation: X ← 00 or,
ACCA ← 00 or,
M ← 00

Description: The contents of ACCA, X or M are replaced with zeroes.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Cleared.
- Z: Set.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = 0$
$Z = 1$

Source Form(s): CLR Q, CLRA, CLRX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	4F
Index Register	4	3	1	5F
Immediate				
Direct	6	5	2	3F
Extended				
Indexed 0 Offset	6	5	1	7F
Indexed 1-Byte	7	6	2	6F
Indexed 2-Byte				

CMP **Compare Accumulator with Memory** CMP

Operation: ACCA − M

Description: Compares the contents of ACCA and the contents of M and sets the condition codes, which may then be used for controlling the conditional branches. Both operands are unaffected.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result of the subtraction is set; cleared otherwise.
- Z: Set if all bits of the result of the subtraction are cleared; cleared otherwise.
- C: Set if the absolute value of the contents of memory is larger than the absolute value of the accumulator; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

$C = \overline{A7} \cdot M7 v M7 \cdot R7 v R7 \cdot \overline{A7}$

Source Form(s): CMP P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A1
Direct	4	3	2	B1
Extended	5	4	3	C1
Indexed 0 Offset	4	3	1	F1
Indexed 1-Byte	5	4	2	E1
Indexed 2-Byte	6	5	3	D1

COM — Complement — COM

Operation: $X \leftarrow \sim X = \$FF - X$ or,
$ACCA \leftarrow \sim ACCA = \$FF - ACCA$ or,
$M \leftarrow \sim M = \$FF - M$

Description: Replaces the contents of ACCA, X or M with the one's complement. Each bit of the operand is replaced with the complement of that bit.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

$C = 1$

Source Form(s): COM Q, COMA, COMX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	43
Index Register	4	3	1	53
Immediate				
Direct	6	5	2	33
Extended				
Indexed 0 Offset	6	5	1	73
Indexed 1-Byte	7	6	2	63
Indexed 2-Byte				

CPX Compare Index Register with Memory CPX

Operation: X − M

Description: Compares the contents of X to the contents of M and sets the condition codes, which may then be used for controlling the conditional branches. Both operands are unaffected.

Condition Codes:

H: Not affected.

I: Not affected.

N: Set if the most significant bit of the result of the subtraction is set; cleared otherwise.

Z: Set if all bits of the result of the subtraction are cleared; cleared otherwise.

C: Set if the absolute value of the contents of memory is larger than the absolute value of the index register; cleared otherwise.

Boolean Formulae for Condition Codes:

$$N = R7$$
$$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$$
$$C = \overline{X7} \cdot M7 \text{v} M7 \cdot \overline{R7} \text{v} \overline{R7} \cdot \overline{X7}$$

Source Form(s): CPX P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A3
Direct	4	3	2	B3
Extended	5	4	3	C3
Indexed 0 Offset	4	3	1	F3
Indexed 1-Byte	5	4	2	E3
Indexed 2-Byte	6	5	3	D3

DEC Decrement DEC

Operation: X ← X-01 or,
ACCA ← ACCA-01 or,
M ← M-01

Description: Subtract one from the contents of ACCA, X or M. The N and Z bits are set or reset according to the result of this operation. The C bit is not affected by this operation.

Condition Codes:
H: Not affected.
I: Not affected.
N: Set if the most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
C: Not affected.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): DEC Q, DECA, DECX, DEX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	4A
Index Register	4	3	1	5A
Immediate				
Direct	6	5	2	3A
Extended				
Indexed 0 Offset	6	5	1	7A
Indexed 1-Byte	7	6	2	6A
Indexed 2-Byte				

EOR — Exclusive Or Memory with Accumulator — EOR

Operation: ACCA ← ACCA ⊕ M

Description: Performs the logical EXCLUSIVE OR between the contents of ACCA and the contents of M, and places the result in ACCA. Each bit of ACCA after the operation will be the logical EXCUSIVE OR of the corresponding bit of M and ACCA before the operation.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): EOR P

Addressing Mode	Cycles		Bytes	Opcode
	HMOS	CMOS		
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A8
Direct	4	3	2	B8
Extended	5	4	3	C8
Indexed 0 Offset	4	3	1	F8
Indexed 1-Byte	5	4	2	E8
Indexed 2-Byte	6	5	3	D8

INC Increment INC

Operation: X ← X + 01 or,
ACCA ← ACCA + 01 or,
M ← M + 01

Description: Add one to the contents of ACCA, X or M. The N and Z bits are set or reset according to the result of this operation. The C bit is not affected by this operation.

Condition Codes:
H: Not affected.
I: Not affected.
N: Set if the most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
C: Not affected.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): INC Q, INCA, INCX, INX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	4C
Index Register	4	3	1	5C
Immediate				
Direct	6	5	2	3C
Extended				
Indexed 0 Offset	6	5	1	7C
Indexed 1-Byte	7	6	2	6C
Indexed 2-Byte				

JMP Jump JMP

Operation: PC ← effective address

Description: A jump occurs to the instruction stored at the effective address. The effective address is obtained according to the rules for EXTended, DIRect or INDexed addressing.

Condition Codes: Not affected.

Source Form(s): JMP P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate				
Direct	3	2	2	BC
Extended	4	3	3	CC
Indexed 0 Offset	3	2	1	FC
Indexed 1-Byte	4	3	2	EC
Indexed 2-Byte	5	4	3	DC

JSR **Jump to Subroutine** JSR

Operation: PC ← PC + N
(SP) ← PCL; SP ← SP − 0001
(SP) ← PCH ; SP ← SP − 0001
PC ← effective address

Description: The program counter is incremented by N (N = 1, 2 or 3 depending on the addressing mode), and is then pushed onto the stack (least significant byte first). Unused bits in the Program Counter high byte are stored as 1's on the stack. The stack pointer points to the next empty location on the stack. A jump occurs to the instruction stored at the effective address. The effective address is obtained according to the rules for EXTended, DIRect, or INDexed addressing.

Condition Codes: Not affected.

Source Form(s): JSR P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate				
Direct	7	5	2	BD
Extended	8	6	3	CD
Indexed 0 Offset	7	5	1	FD
Indexed 1-Byte	8	6	2	ED
Indexed 2-Byte	9	7	3	DD

LDA — Load Accumulator from Memory — LDA

Operation: ACCA ← M

Description: Loads the contents of memory into the accumulator. The condition codes are set according to the data.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the accumulator is set; cleared otherwise.
- Z: Set if all bits of the accumulator are cleared; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): LDA P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A6
Direct	4	3	2	B6
Extended	5	4	3	C6
Indexed 0 Offset	4	3	1	F6
Indexed 1-Byte	5	4	2	E6
Indexed 2-Byte	6	5	3	D6

LDX — Load Index Register from Memory — LDX

Operation: $X \leftarrow M$

Description: Loads the contents of memory into the index register. The condition codes are set according to the data.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the index register is set; cleared otherwise.
- Z: Set if all bits of the index register are cleared; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$$N = R7$$
$$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$$

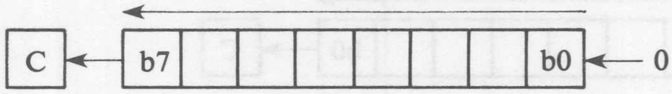
Source Form(s): LDX P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	AE
Direct	4	3	2	BE
Extended	5	4	3	CE
Indexed 0 Offset	4	3	1	FE
Indexed 1-Byte	5	4	2	EE
Indexed 2-Byte	6	5	3	DE

LSL

Logical Shift Left

Operation:



Description: Shifts all bits of the ACCA, X or M one place to the left. Bit 0 is loaded with a zero. The C bit is loaded from the most signficant bit of ACCA, X or M.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if the most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
C: Set if, before the operation, the most significant bit of ACCA, X or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$
$Z = \overline{R7}\bullet\overline{R6}\bullet\overline{R5}\bullet\overline{R4}\bullet\overline{R3}\bullet\overline{R2}\bullet\overline{R1}\bullet\overline{R0}$
$C = M7$

Comments: Same as ASL

Source Form(s): LSL Q, LSLA, LSLX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	48
Index Register	4	3	1	58
Immediate				
Direct	6	5	2	38
Extended				
Indexed 0 Offset	6	5	1	78
Indexed 1-Byte	7	6	2	68
Indexed 2-Byte				

LSR — Logical Shift Right — LSR

Operation: 0 → | b7 | | | | | | | b0 | → | C |

Description: Shifts all bits of ACCA, X or M one place to the right. Bit 7 is loaded with a zero. Bit 0 is loaded into the C bit.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Cleared.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if, before the operation, the least significant bit of ACCA, X or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = 0$

$Z = \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$

$C = M0$

Source Form(s): LSR Q, LSRA, LSRX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	44
Index Register	4	3	1	54
Immediate				
Direct	6	5	2	34
Extended				
Indexed 0 Offset	6	5	1	74
Indexed 1-Byte	7	6	2	64
Indexed 2-Byte				

NEG Negate NEG

Operation: $-X \rightarrow X = 00 - X$ or,
$-ACCA \rightarrow ACCA = 00 - ACCA$ or,
$-M \rightarrow M = 00 - M$

Description: Replaces the contents of ACCA, X or M with its two's complement. Note that \$80 is left unchanged.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if the most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
C: Set if there would be a borrow in the implied subtraction from zero; the C bit will be set in all cases except when the contents of ACCA, X or M before the NEG is 00.

Boolean Formulae for Condition Codes:

$$N = R7$$
$$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$$
$$C = R7 v R6 v R5 v R4 v R3 v R2 v R1 v R0$$

Source Form(s): NEG Q, NEGA, NEGX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	40
Index Register	4	3	1	50
Immediate				
Direct	6	5	2	30
Extended				
Indexed 0 Offset	6	5	1	70
Indexed 1-Byte	7	6	2	60
Indexed 2-Byte				

NOP No Operation NOP

Description: This is a single-byte instruction which causes only the program counter to be incremented. No other registers are changed.

Condition Codes: Not affected.

Source Form(s): NOP

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	9D
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

ORA Inclusive OR ORA

Operation: ACCA ← ACCA V M

Description: Performs logical OR between the contents of ACCA and the contents of M and place the result in ACCA. Each bit of ACCA after the operation will be the logical (inclusive) OR result of the corresponding bits of M and ACCA before the operation.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if the most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
C: Not affected.

Boolean Formulae for Condition Codes:

$N = R7$
$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

Source Form(s): ORA P

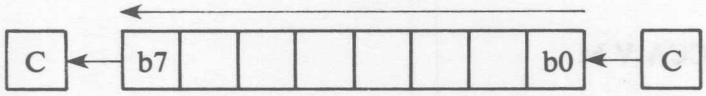
Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	AA
Direct	4	3	2	BA
Extended	5	4	3	CA
Indexed 0 Offset	4	3	1	FA
Indexed 1-Byte	5	4	2	EA
Indexed 2-Byte	6	5	3	DA

 ROL

ROL

Rotate Left thru Carry

Operation:



Description: Shifts all bits of the ACCA, X or M one place to the left. Bit 0 is loaded from the C bit. The C bit is loaded from the most significant bit of ACCA, X or M.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if, before the operation, the most significant bit of ACCA, X or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

$C = M7$

Source Form(s): ROL Q, ROLA, ROLX

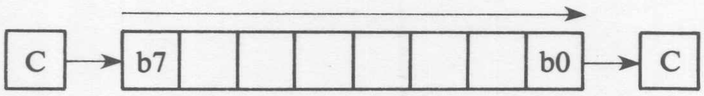
Addressing Mode	**Cycles HMOS**	**Cycles CMOS**	**Bytes**	**Opcode**
Inherent				
Relative				
Accumulator	4	3	1	49
Index Register	4	3	1	59
Immediate				
Direct	6	5	2	39
Extended				
Indexed 0 Offset	6	5	1	79
Indexed 1-Byte	7	6	2	69
Indexed 2-Byte				

ROR

ROR

Rotate Right Thru Carry

Operation:



Description: Shifts all bits of ACCA, X or M one place to the right. Bit 7 is loaded from the C bit. Bit 0 is loaded into the C bit.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the result are cleared; cleared otherwise.
- C: Set if, before the operation, the least significant bit of ACCA, X or M was set; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

$C = M0$

Source Form(s): ROR Q, RORA, RORX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	46
Index Register	4	3	1	56
Immediate				
Direct	6	5	2	36
Extended				
Indexed 0 Offset	6	5	1	76
Indexed 1-Byte	7	6	2	66
Indexed 2-Byte				

RSP

Reset Stack Pointer

Operation: SP ← \$7F

Description: Resets the stack pointer to the top of the stack.

Condition Codes: Not affected.

Source Form(s): RSP

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	9C
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

RTI

Return from Interrupt

RTI

Operation:

SP ← SP + 0001 ; CC ← (SP)
SP ← SP + 0001 ; ACCA ← (SP)
SP ← SP + 0001 ; X ← (SP)
SP ← SP + 0001 ; PCH ← (SP)
SP ← SP + 0001 ; PCL ← (SP)

Description: The Condition Codes, Accumulator, Index Register and the Program Counter are restored according to the state previously saved on the stack. Note that the interrupt mask bit (I bit) will be reset if and only if the corresponding bit stored on the stack is zero.

Condition Codes: Set or cleared according to the first byte pulled from the stack.

Source Form(s): RTI

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	9	9	1	80
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

RTS

Return from Subroutine

Operation: SP ← SP + 0001 ; PCH ← (SP)
SP ← SP + 0001 ; PCL ← (SP)

Description: The stack pointer is incremented (by one). The contents of the byte of memory, pointed to by the stack pointer, are loaded into the high byte of the program counter. The stack pointer is again incremented (by one). The byte pointed to by the stack pointer is loaded into the low byte of the program counter.

Condition Codes: Not affected.

Source Form(s): RTS

Addressing Mode	Cycles HMOS	CMOS	Bytes	Opcode
Inherent	6	6	1	81
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

SBC Subtract with Carry SBC

Operation: ACCA ← ACCA − M − C

Description: Subtracts the contents of M and C from the contents of ACCA, and places the result in ACCA.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if the most significant bit of the result is set; cleared otherwise.
Z: Set if all bits of the result are cleared; cleared otherwise.
C: Set if the absolute value of the contents of memory plus the previous carry is larger than the absolute value of the accumulator; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$
$Z = \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
$C = \overline{A7} \bullet M7 \vee M7 \bullet R7 \vee R7 \bullet \overline{A7}$

Source Form(s): SBC P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A2
Direct	4	3	2	B2
Extended	5	4	3	C2
Indexed 0 Offset	4	3	1	F2
Indexed 1-Byte	5	4	2	E2
Indexed 2-Byte	6	5	3	D2

SEC — Set Carry Bit — SEC

Operation: C bit ← 1

Description: Sets the carry bit in the processor condition code register.

Condition Codes:

H: Not affected.
I: Not affected.
N: Not affected.
Z: Not affected.
C: Set.

Boolean Formulae for Condition Codes:

C = 1

Source Form(s): SEC

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	99
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

SEI — Set Interrupt Mask Bit — SEI

Operation: I bit ← 1

Description: Sets the interrupt mask bit in the processor condition code register. The microprocessor is inhibited from servicing interrupts, and will continue with execution of the instructions of the program until the interrupt mask bit is cleared.

Condition Codes:
H: Not affected.
I: Set
N: Not Affected.
Z: Not affected.
C: Not affected.

Boolean Formulae for Condition Codes:
I = 1

Source Form(s): SEI

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	9B
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

STA — Store Accumulator in Memory — STA

Operation: M ← ACCA

Description: Stores the contents of ACCA in memory. The contents of ACCA remain the same.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if the most significant bit of the accumulator is set; cleared otherwise.
Z: Set if all bits of the accumulator are clear; cleared otherwise.
C: Not Affected.

Boolean Formulae for Condition Codes:

$N = A7$
$Z = \overline{A7} \cdot \overline{A6} \cdot \overline{A5} \cdot \overline{A4} \cdot \overline{A3} \cdot \overline{A2} \cdot \overline{A1} \cdot \overline{A0}$

Source Form(s): STA P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate				
Direct	5	4	2	B7
Extended	6	5	3	C7
Indexed 0 Offset	5	4	1	F7
Indexed 1-Byte	6	5	2	E7
Indexed 2-Byte	7	6	3	D7

STOP — Enable IRQ, Stop Oscillator — STOP

Description: Reduces power consumption by eliminating all dynamic power dissipation. Results in: (1) timer prescaler to clear; (2) disabling of timer interrupts (3) timer interrupt flag bit to clear; (4) external interrupt request enabling; and (5) inhibiting of oscillator.

When $\overline{\text{RESET}}$ or $\overline{\text{IRQ}}$ input goes low: (1) oscillator is enabled, (2) a delay of 1920 instruction cycles allows oscillator to stabilize, (3) the interrupt request vector is fetched, and (4) service routine is executed.

External interrupts are enabled following the RTI command.

Condition Codes:

H: Not Affected.
I: Cleared.
N: Not Affected.
Z: Not Affected.
C: Not Affected.

Source Form(s): STOP

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	—	2	1	8E
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

STX — Store Index Register in Memory — STX

Operation: M ← X

Description: Stores the contents of X in memory. The contents of X remain the same.

Condition Codes:

- H: Not Affected.
- I: Not affected.
- N: Set if the most significant bit of the index register is set; cleared otherwise.
- Z: Set if all bits of the index register are clear; cleared otherwise.
- C: Not affected.

Boolean Formulae for Condition Codes:

$N = X7$

$Z = \overline{X7} \cdot \overline{X6} \cdot \overline{X5} \cdot \overline{X4} \cdot \overline{X3} \cdot \overline{X2} \cdot \overline{X1} \cdot \overline{X0}$

Source Form(s): STX P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate				
Direct	5	4	2	BF
Extended	6	5	3	CF
Indexed 0 Offset	5	4	1	FF
Indexed 1-Byte	6	5	2	EF
Indexed 2-Byte	7	6	3	DF

SUB — Subtract — SUB

Operation: ACCA ← ACCA − M

Description: Subtracts the contents of M from the contents of ACCA and places the result in ACCA.

Condition Codes:

- H: Not affected.
- I: Not affected.
- N: Set if the most significant bit of the result is set; cleared otherwise.
- Z: Set if all bits of the results are cleared; cleared otherwise.
- C: Set if the absolute value of the contents of memory are larger than the absolute value of the accumulator; cleared otherwise.

Boolean Formulae for Condition Codes:

$N = R7$

$Z = \overline{R7} \cdot \overline{R6} \cdot \overline{R5} \cdot \overline{R4} \cdot \overline{R3} \cdot \overline{R2} \cdot \overline{R1} \cdot \overline{R0}$

$C = A7 \bullet M7 \vee M7 \bullet \overline{R7} \vee \overline{R7} \bullet A7$

Source Form(s): SUB P

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator				
Index Register				
Immediate	2	2	2	A0
Direct	4	3	2	B0
Extended	5	4	3	C0
Indexed 0 Offset	4	3	1	F0
Indexed 1-Byte	5	4	2	E0
Indexed 2-Byte	6	5	3	D0

SWI Software Interrupt SWI

Operation:

PC ← PC + 0001
(SP) ← PCL ; SP ← SP − 0001
(SP) ← PCH ; SP ← SP − 0001
(SP) ← X ; SP ← SP − 0001
(SP) ← ACCA ; SP ← SP − 0001
(SP) ← CC ; SP ← SP − 0001
I bit ← 1
PCH ← n − 0003
PCL ← n − 0002

Description: The program counter is incremented (by one). The Program Counter, Index Register and Accumulator are pushed onto the stack. The Condition Code register bits are then pushed onto the stack with bits H, I, N, Z and C going into bit positions 4 through 0 with the top three bits (7, 6 and 5) containing ones. The stack pointer is decremented by one after each byte is stored on the stack.

The interrupt mask bit is then set. The program counter is then loaded with the address stored in the software interrupt vector located at memory locations n − 0002 and n − 0003, where n is the address corresponding to a high state on all lines of the address bus.

Condition Codes:

H: Not affected.
I: Set.
N: Not affected.
Z: Not affected.
C: Not affected.

Boolean Formulae for Condition Codes:

I = 1

Caution: This instruction is used by Motorola in some of its software products and may be unavailable for general use.

Source Form(s): SWI

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	11	10	1	83
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

TAX Transfer Accumulator to Index Register TAX

Operation: X ← ACCA

Description: Loads the index register with the contents of the accumulator. The contents of the accumulator are unchanged.

Condition Codes: Not affected.

Source Form(s): TAX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	97
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

TST — Test for Negative or Zero — TST

Operation: X − 00 or,
ACCA − 00 or,
M − 0

Description: Sets the condition codes N and Z according to the contents of ACCA, X or M.

Condition Codes:

H: Not affected.
I: Not affected.
N: Set if the most significant bit of the contents of ACCA, X or M is set; cleared otherwise.
Z: Set if all bits of ACCA, X or M are clear; cleared otherwise.
C: Not affected.

Boolean Formulae for Condition Codes:

$N = M7$
$Z = \overline{M7} \bullet \overline{M6} \bullet \overline{M5} \bullet \overline{M4} \bullet \overline{M3} \bullet \overline{M2} \bullet \overline{M1} \bullet \overline{M0}$

Source Form(s): TST Q, TSTA, TSTX

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent				
Relative				
Accumulator	4	3	1	4D
Index Register	4	3	1	5D
Immediate				
Direct	6	4	2	3D
Extended				
Indexed 0 Offset	6	4	1	7D
Indexed 1-Byte	7	5	2	6D
Indexed 2-Byte				

TXA — Transfer Index Register to Accumulator — TXA

Operation: ACCA ← X

Description: Loads the accumulator with the contents of the index register. The contents of the index register are unchanged.

Condition Codes: Not affected.

Source Form(s): TXA

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	2	2	1	9F
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

WAIT **Enable Interrupt, Stop Processor** WAIT

Description: Reduces power consumption by eliminating dynamic power dissipation in all circuits except the timer and timer prescaler. Causes enabling of external interrupts and stops clocking or processor circuits.

Timer interrupts may be enabled or disabled by programmer prior to execution of WAIT.

When RESET or IRQ input goes low, or timer counter reaches zero with counter interrupt enabled: (1) processor clocks are enabled, and (2) interrupt request, reset, and timer interrupt vectors are fetched.

Interrupts are enabled following the RTI command.

Condition Codes:

H: Not affected.
I: Cleared.
N: Not affected.
Z: Not affected.
C: Not affected.

Source Form(s): WAIT

Addressing Mode	Cycles HMOS	Cycles CMOS	Bytes	Opcode
Inherent	—	2	1	8F
Relative				
Accumulator				
Index Register				
Immediate				
Direct				
Extended				
Indexed 0 Offset				
Indexed 1-Byte				
Indexed 2-Byte				

APPENDIX E
INSTRUCTION SET
ALPHABETICAL LISTING

This appendix provides an alphabetical listing of the Mnemonic Instruction Set, together with Addressing Modes used and the effects on the condition code register.

	Addressing Modes										Condition Codes				
Mnemonic	Inherent	Immediate	Direct	Extended	Relative	Indexed (No Offset)	Indexed (8 Bits)	Indexed (16 Bits)	Bit Set/ Clear	Bit Test & Branch.	H	I	N	Z	C
ADC		X	X	X		X	X	X			Λ	•	Λ	Λ	Λ
ADD		X	X	X		X	X	X			Λ	•	Λ	Λ	Λ
AND		X	X	X		X	X	X			•	•	Λ	Λ	•
ASL	X		X			X	X				•	•	Λ	Λ	Λ
ASR	X		X			X	X				•	•	Λ	Λ	Λ
BCC					X						•	•	•	•	•
BCLR									X		•	•	•	•	•
BCS					X						•	•	•	•	•
BEQ					X						•	•	•	•	•
BHCC					X						•	•	•	•	•
BHCS					X						•	•	•	•	•
BHI					X						•	•	•	•	•
BHS					X						•	•	•	•	•
BIH					X						•	•	•	•	•
BIL					X						•	•	•	•	•
BIT		X	X	X		X	X	X			•	•	Λ	Λ	•
BLO					X						•	•	•	•	•
BLS					X						•	•	•	•	•
BMC					X						•	•	•	•	•
BMI					X						•	•	•	•	•
BMS					X						•	•	•	•	•
BNE					X						•	•	•	•	•
BPL					X						•	•	•	•	•
BRA					X						•	•	•	•	•
BRN					X						•	•	•	•	•
BRCLR										X	•	•	•	•	Λ
BRSET										X	•	•	•	•	Λ
BSET									X		•	•	•	•	•
BSR					X						•	•	•	•	•
CLC	X										•	•	•	•	0
CLI	X										•	0	•	•	•
CLR	X		X			X	X				•	•	0	1	•
CMP		X	X	X		X	X	X			•	•	Λ	Λ	Λ
COM	X		X			X	X				•	•	Λ	Λ	1
CPX		X	X	X		X	X	X			•	•	Λ	Λ	Λ

Mnemonic	Addressing Modes										Condition Codes				
	Inherent	Immediate	Direct	Extended	Relative	Indexed (No Offset)	Indexed (8 Bits)	Indexed (16 Bits)	Bit Set/ Clear	Bit Test & Branch	H	I	N	Z	C
DEC	X		X			X	X				•	•	Λ	Λ	•
EOR		X	X	X		X	X	X			•	•	Λ	Λ	•
INC	X		X			X	X				•	•	Λ	Λ	•
JMP			X	X		X	X	X			•	•	•	•	•
JSR			X	X		X	X	X			•	•	•	•	•
LDA		X	X	X		X	X	X			•	•	Λ	Λ	•
LDX		X	X	X		X	X	X			•	•	Λ	Λ	•
LSL	X		X			X	X				•	•	Λ	Λ	Λ
LSR	X		X			X	X				•	•	0	Λ	Λ
NEQ	X		X			X	X				•	•	Λ	Λ	Λ
NOP	X										•	•	•	•	•
ORA		X	X	X		X	X	X			•	•	Λ	Λ	•
ROL	X		X			X	X				•	•	Λ	Λ	Λ
RSP	X										•	•	•	•	•
RTI	X										?	?	?	?	?
RTS	X										•	•	•	•	•
SBC		X	X	X		X	X	X			•	•	Λ	Λ	Λ
SEC	X										•	•	•	•	1
SEI	X										•	1	•	•	•
STA			X	X		X	X	X			•	•	Λ	Λ	•
STX			X	X		X	X	X			•	•	Λ	Λ	•
STOP	X										•	1	•	•	•
SUB		X	X	X		X	X	X			•	•	Λ	Λ	Λ
SWI	X										•	1	•	•	•
TAX	X										•	•	•	•	•
TST	X		X			X	X				•	•	Λ	Λ	•
TXA	X										•	•	•	•	•
WAIT	X										•	1	•	•	•

Condition Code Symbols

H Half Carry (From Bit 3)
I Interrupt Mask
N Negative (Sign Bit)
Z Zero
C Carry/Borrow
Λ Test and Set if True, Cleared Otherwise
• Not Affected
? Load CC Register From Stack

APPENDIX F
INSTRUCTION SET
FUNCTIONAL LISTING

This Instruction Set contains a list of functions which are categorized as to the type of instruction. It provides five different categories of instructions and provides the following information for each function: (1) Corresponding Mnemonic, (2) Addressing Mode, (3) Op Code, (4) Number of Bytes, and (5) number of cycles.

Branch Instructions

Function	Mnemonic	Relative Addressing Mode		
		Op Code	# Bytes	HMOS/CMOS # Of Cycles
Branch Always	BRA	20	2	4/3
Branch Never	BRN	21	2	4/3
Branch IFF Higher	BHI	22	2	4/3
Branch IFF Lower or Same	BLS	23	2	4/3
Branch IFF Carry Clear	BCC	24	2	4/3
(Branch IFF Higher or Same)	(BHS)	24	2	4/3
Branch IFF Carry Set	BCS	25	2	4/3
(Branch IFF Lower)	(BLO)	25	2	4/3
Branch IFF Not Equal	BNE	26	2	4/3
Branch IFF Equal	BEQ	27	2	4/3
Branch IFF Half Carry Clear	BHCC	28	2	4/3
Branch IFF Half Carry Set	BHCS	29	2	4/3
Branch IFF Plus	BPL	2A	2	4/3
Branch IFF Minus	BMI	2B	2	4/3
Branch IFF Interrupt Mask Bit is Clear	BMC	2C	2	4/3
Branch IFF Interrupt Mask Bit is Set	BMS	2D	2	4/3
Branch IFF Interrupt Line is Low	BIL	2E	2	4/3
Branch IFF Interrupt Line is High	BIH	2F	2	4/3
Branch to Subroutine	BSR	AD	2	8/6

Bit Manipulation Instructions

Function	Mnemonic	Addressing Modes					
		Bit Set/Clear			Bit Test and Branch		
		Op Code	# Bytes	HMOS/CMOS # of Cycles	Op Code	# Bytes	HMOS/CMOS # of Cycles
Branch IFF Bit n is set	BRSET n (n = 0....7)	—	—	—	2 • n	3	10/5
Branch IFF Bit n is clear	BRCLR n (n = 0....7)	—	—	—	01 + 2 • n	3	10/5
Set Bit n	BSET n (n = 0....7)	10 + 2 • n	2	7/5	—	—	—
Clear bit n	BCLR n (n = 0....7)	11 + 2 • n	2	7/5	—	—	—

Control Instructions

Function	Mnemonic	Inherent		
		Op Code	# Bytes	HMOS/CMOS # of Cycles
Transfer A to X	TAX	97	1	2/2
Transfer X to A	TXA	9F	1	2/2
Set Carry Bit	SEC	99	1	2/2
Clear Carry Bit	CLC	98	1	2/2
Set Interrupt Mask Bit	SEI	9B	1	2/2
Clear Interrupt Mask Bit	CLI	9A	1	2/2
Software Interrupt	SWI	83	1	11/10
Return from Subroutine	RTS	81	1	6/6
Return from Interrupt	RTI	80	1	9/9
Reset Stack Pointer	RSP	9C	1	2/2
No-Operation	NOP	9D	1	2/2
Enable IRQ, Stop Oscillator	STOP	8E	1	-/2
Enable Interrupt, Stop Processor	WAIT	8F	1	-/2

Read/Modify/Write Instructions

Function	Mnem.	Inherent (A) Op Code	Inherent (A) # Bytes	Inherent (A) Cycles (see note)	Inherent (X) Op Code	Inherent (X) # Bytes	Inherent (X) Cycles (see note)	Direct Op Code	Direct # Bytes	Direct Cycles (see note)	Indexed (No Offset) Op Code	Indexed (No Offset) # Bytes	Indexed (No Offset) Cycles (see note)	Indexed (8-Bit Offset) Op Code	Indexed (8-Bit Offset) # Bytes	Indexed (8-Bit Offset) Cycles (see note)
Increment	INC	4C	1	4/3	5C	1	4/3	3C	2	6/5	7C	1	6/5	6C	2	7/6
Decrement	DEC	4A	1	4/3	5A	1	4/3	3A	2	6/5	7A	1	6/5	6A	2	7/6
Clear	CLR	4F	1	4/3	5F	1	4/3	3F	2	6/5	7F	1	6/5	6F	2	7/6
Complement	COM	43	1	4/3	53	1	4/3	33	2	6/5	73	1	6/5	63	2	7/6
Negate (2's complement)	NEG	40	1	4/3	50	1	4/3	30	2	6/5	70	1	6/5	60	2	7/6
Rotate Left Thru Carry	ROL	49	1	4/3	59	1	4/3	39	2	6/5	79	1	6/5	69	2	7/6
Rotate Right Thru Carry	ROR	46	1	4/3	56	1	4/3	36	2	6/5	76	1	6/5	66	2	7/6
Logical Shift Left	LSL	48	1	4/3	58	1	4/3	38	2	6/5	78	1	6/5	68	2	7/6
Logical Shift Right	LSR	44	1	4/3	54	1	4/3	34	2	6/5	74	1	6/5	64	2	7/6
Arithmetic Shift Right	ASR	47	1	4/3	57	1	4/3	37	2	6/5	77	1	6/5	67	2	7/6
Test for Negative or Zero	TST	4D	1	4/3	5D	1	4/3	3D	2	6/4	7D	1	6/4	6D	2	7/5

NOTE: The cycles column actually shows the number of HMOS/CMOS cycles (e.g., 4/3 indicates 4 HMOS cycles or 3 CMOS cycles).

Register/Memory Instructions

Function	Mnem.	Immediate			Direct			Extended			Indexed (No Offset)			Indexed (8-Bit Offset)			Indexed (16-Bit Offset)		
		Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)	Op Code	# Bytes	Cycles (see note)
Load A from Memory	LDA	A6	2	2/2	B6	2	4/3	C6	3	5/4	F6	1	4/3	E6	2	5/4	D6	3	6/5
Load X from Memory	LDX	AE	2	2/2	BE	2	4/3	CE	3	5/4	FE	1	4/3	EE	2	5/4	DE	3	6/5
Store A in Memory	STA	–	–	–	B7	2	5/4	C7	3	6/5	F7	1	5/4	E7	2	6/5	D7	3	7/6
Store X in Memory	STX	–	–	–	BF	2	5/4	CF	3	6/5	FF	1	5/4	EF	2	6/5	DF	3	7/6
Add Memory to A	ADD	AB	2	2/2	BB	2	4/3	CB	3	5/4	FB	1	4/3	EB	2	5/4	DB	3	6/5
Add Memory and Carry to A	ADC	A9	2	2/2	B9	2	4/3	C9	3	5/4	F9	1	4/3	E9	2	5/4	D9	3	6/5
Subtract Memory	SUB	A0	2	2/2	B0	2	4/3	C0	3	5/4	F0	1	4/3	E0	2	5/4	D0	3	6/5
Subtract Memory from A with Borrow	SBC	A2	2	2/2	B2	2	4/3	C2	3	5/4	F2	1	4/3	E2	2	5/4	D2	3	6/5
AND Memory to A	AND	A4	2	2/2	B4	2	4/3	C4	3	5/4	F4	1	4/3	E4	2	5/4	D4	3	6/5
OR Memory with A	ORA	AA	2	2/2	BA	2	4/3	CA	3	5/4	FA	1	4/3	EA	2	5/4	DA	3	6/5
Exclusive OR Memory with A	EOR	A8	2	2/2	B8	2	4/3	C8	3	5/4	F8	1	4/3	E8	2	5/4	D8	3	6/5
Arithmetic Compare A with Memory	CMP	A1	2	2/2	B1	2	4/3	C1	3	5/4	F1	1	4/3	F1	2	5/4	D1	3	6/5
Arithmetic Compare X with Memory	CPX	A3	2	2/2	B3	2	4/3	C3	3	5/4	F3	1	4/3	E3	2	5/4	D3	3	6/5
Bit Test Memory with A (Logical Compare)	BIT	A5	2	2/2	B5	2	4/3	C5	3	5/4	F5	1	4/3	E5	2	5/4	D5	3	6/5
Jump Unconditional	JMP	–	–	–	BC	2	3/2	CC	3	4/3	FC	1	3/2	EC	2	4/3	DC	3	5/4
Jump to Subroutine	JSR	–	–	–	BD	2	7/5	CD	3	8/6	FD	1	7/5	ED	2	8/6	DD	3	9/7

NOTE: The cycles column actually shows the number of HMOS/CMOS cycles (e.g., 4/3 indicates 4 HMOS cycles or 3 CMOS cycles).

APPENDIX G
ASCII HEXADECIMAL CODE CONVERSION CHART

This appendix shows the equivalent alphanumeric characters for the equivalent ASCII hexadecimal code.

Hex	ASCII	Hex	ASCII	Hex	ASCII	Hex	ASCII
00	nul	20	sp	40	@	60	'
01	soh	21	!	41	A	61	a
02	stx	22	"	42	B	62	b
03	etx	23	#	43	C	63	c
04	eot	24	$	44	D	64	d
05	enq	25	%	45	E	65	e
06	ack	26	&	46	F	66	f
07	bel	27	'	47	G	67	g
08	bs	28	(	48	H	68	h
09	ht	29	)	49	I	69	i
0A	nl	2A	*	4A	J	6A	j
0B	vt	2B	+	4B	K	6B	k
0C	ff	2C	,	4C	L	6C	l
0D	cr	2D	-	4D	M	6D	m
0E	so	2E	.	4E	N	6E	n
0F	si	2F	/	4F	O	6F	o
10	dle	30	0	50	P	70	p
11	dc1	31	1	51	Q	71	q
12	dc2	32	2	52	R	72	r
13	dc3	33	3	53	S	73	s
14	dc4	34	4	54	T	74	t
15	nak	35	5	55	U	75	u
16	syn	36	6	56	V	76	v
17	etb	37	7	57	W	77	w
18	can	38	8	58	X	78	x
19	em	39	9	59	Y	79	y
1A	sub	3A	:	5A	Z	7A	z
1B	esc	3B	;	5B	[	7B	{
1C	fs	3C	<	5C	\	7C	¦
1D	gs	3D	=	5D	]	7D	}
1E	rs	3E	>	5E	Λ	7E	~
1F	us	3F	?	5F	—	7F	del

APPENDIX H
INSTRUCTION SET
OPCODE MAP

The Opcode Map contains a summary of opcodes used with the M6805 and M146805 Family. The map is outlined by two sets (0-F) of hexadecimal numbers; one horizontal and one vertical. The horizontal set represents the MSD and the vertical set represents the LSD. For example, a 25 opcode represents a BCS (located at the 2 and 5 coordinates) used in the Relative Mode. There are five different opcodes for COM, each in a different addressing mode (Direct; Accumulator; Indexed; Indexed, one byte offset; and Indexed, two byte offset). A legend is provided, as part of the map, to show the information contained in each coordinate square. The legend represents the coordinates for Opcode F0 (SUB). Included in the legend is the opcode binary equivalent, the number of execution cycles required for both the M6805 (HMOS) and M146805 (CMOS) Family, the required number of bytes, the address mode, and the mnemonic.

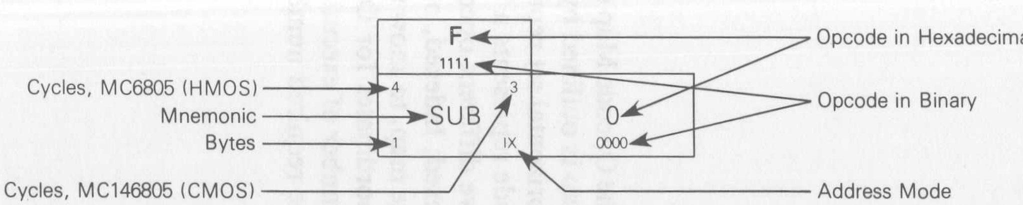
MC6805/MC146805 Instruction Set Opcode Map

	Bit Manipulation		Branch	Read/Modify/Write					Control		Register/Memory						
	BTB	BSC	REL	DIR	A	X	IX1	IX	INH	INH	IMM	DIR	EXT	IX2	IX1	IX	
Low \ Hi	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	Hi \ Low
0 0000	10 5 BRSET0 3 BTB	7 5 BSET0 2 BSC	4 3 BRA 2 REL	6 5 NEG 2 DIR	4 3 NEG 1 A	4 3 NEG 1 X	7 6 NEG 2 IX1	6 5 NEG 1 IX	9 9 RTI 1 INH		2 2 SUB 2 IMM	4 3 SUB 2 DIR	5 4 SUB 3 EXT	6 5 SUB 3 IX2	5 4 SUB 2 IX1	4 3 SUB 1 IX	0 0000
1 0001	10 5 BRCLR0 3 BTB	7 5 BCLR0 2 BSC	4 3 BRN 2 REL						6 6 RTS 1 INH		2 2 CMP 2 IMM	4 3 CMP 2 DIR	5 4 CMP 3 EXT	6 5 CMP 3 IX2	5 4 CMP 2 IX1	4 3 CMP 1 IX	1 0001
2 0010	10 5 BRSET1 3 BTB	7 5 BSET1 2 BSC	4 3 BHI 2 REL								2 2 SBC 2 IMM	4 3 SBC 2 DIR	5 4 SBC 3 EXT	6 5 SBC 3 IX2	5 4 SBC 2 IX1	4 3 SBC 1 IX	2 0010
3 0011	10 5 BRCLR1 3 BTB	7 5 BCLR1 2 BSC	4 3 BLS 2 REL	6 5 COM 2 DIR	4 3 COM 1 A	4 3 COM 1 X	7 6 COM 2 IX1	6 5 COM 1 IX	11 10 SWI 1 INH		2 2 CPX 2 IMM	4 3 CPX 2 DIR	5 4 CPX 3 EXT	6 5 CPX 3 IX2	5 4 CPX 2 IX1	4 3 CPX 1 IX	3 0011
4 0100	10 5 BRSET2 3 BTB	7 5 BSET2 2 BSC	4 3 BCC 2 REL	6 5 LSR 2 DIR	4 3 LSR 1 A	4 3 LSR 1 X	7 6 LSR 2 IX1	6 5 LSR 1 IX			2 2 AND 2 IMM	4 3 AND 2 DIR	5 4 AND 3 EXT	6 5 AND 3 IX2	5 4 AND 2 IX1	4 3 AND 1 IX	4 0100
5 0101	10 5 BRCLR2 3 BTB	7 5 BCLR2 2 BSC	4 3 BCS 2 REL								2 2 BIT 2 IMM	4 3 BIT 2 DIR	5 4 BIT 3 EXT	6 5 BIT 3 IX2	5 4 BIT 2 IX1	4 3 BIT 1 IX	5 0101
6 0110	10 5 BRSET3 3 BTB	7 5 BSET3 2 BSC	4 3 BNE 2 REL	6 5 ROR 2 DIR	4 3 ROR 1 A	4 3 ROR 1 X	7 6 ROR 2 IX1	6 5 ROR 1 IX			2 2 LDA 2 IMM	4 3 LDA 2 DIR	5 4 LDA 3 EXT	6 5 LDA 3 IX2	5 4 LDA 2 IX1	4 3 LDA 1 IX	6 0110
7 0111	10 5 BRCLR3 3 BTB	7 5 BCLR3 2 BSC	4 3 BEQ 2 REL	6 5 ASR 2 DIR	4 3 ASR 1 A	4 3 ASR 1 X	7 6 ASR 2 IX1	6 5 ASR 1 IX		2 2 TAX 1 INH		5 4 STA 2 DIR	6 5 STA 3 EXT	7 6 STA 3 IX2	6 5 STA 2 IX1	5 4 STA 1 IX	7 0111
8 1000	10 5 BRSET4 3 BTB	7 5 BSET4 2 BSC	4 3 BHCC 2 REL	6 5 LSL 2 DIR	4 3 LSL 1 A	4 3 LSL 1 X	7 6 LSL 2 IX1	6 5 LSL 1 IX		2 2 CLC 1 INH	2 2 EOR 2 IMM	4 3 EOR 2 DIR	5 4 EOR 3 EXT	6 5 EOR 3 IX2	5 4 EOR 2 IX1	4 3 EOR 1 IX	8 1000
9 1001	10 5 BRCLR4 3 BTB	7 5 BCLR4 2 BSC	4 3 BHCS 2 REL	6 5 ROL 2 DIR	4 3 ROL 1 A	4 3 ROL 1 X	7 6 ROL 2 IX1	6 5 ROL 1 IX		2 2 SEC 1 INH	2 2 ADC 2 IMM	4 3 ADC 2 DIR	5 4 ADC 3 EXT	6 5 ADC 3 IX2	5 4 ADC 2 IX1	4 3 ADC 1 IX	9 1001
A 1010	10 5 BRSET5 3 BTB	7 5 BSET5 2 BSC	4 3 BPL 2 REL	6 5 DEC 2 DIR	4 3 DEC 1 A	4 3 DEC 1 X	7 6 DEC 2 IX1	6 5 DEC 1 IX		2 2 CLI 1 INH	2 2 ORA 2 IMM	4 3 ORA 2 DIR	5 4 ORA 3 EXT	6 5 ORA 3 IX2	5 4 ORA 2 IX1	4 3 ORA 1 IX	A 1010
B 1011	10 5 BRCLR5 3 BTB	7 5 BCLR5 2 BSC	4 3 BMI 2 REL							2 2 SEI 1 INH	2 2 ADD 2 IMM	4 3 ADD 2 DIR	5 4 ADD 3 EXT	6 5 ADD 3 IX2	5 4 ADD 2 IX1	4 3 ADD 1 IX	B 1011
C 1100	10 5 BRSET6 3 BTB	7 5 BSET6 2 BSC	4 3 BMC 2 REL	6 5 INC 2 DIR	4 3 INC 1 A	4 3 INC 1 X	7 6 INC 2 IX1	6 5 INC 1 IX		2 2 RSP 1 INH		3 2 JMP 2 DIR	4 3 JMP 3 EXT	5 4 JMP 3 IX2	4 3 JMP 2 IX1	3 2 JMP 1 IX	C 1100
D 1101	10 5 BRCLR6 3 BTB	7 5 BCLR6 2 BSC	4 3 BMS 2 REL	6 4 TST 2 DIR	4 3 TST 1 A	4 3 TST 1 X	7 5 TST 2 IX1	6 4 TST 1 IX		2 2 NOP 1 INH	8 6 BSR 2 REL	7 5 JSR 2 DIR	8 6 JSR 3 EXT	9 7 JSR 3 IX2	8 6 JSR 2 IX1	7 5 JSR 1 IX	D 1101
E 1110	10 5 BRSET7 3 BTB	7 5 BSET7 2 BSC	4 3 BIL 2 REL						2 STOP 1 INH		2 2 LDX 2 IMM	4 3 LDX 2 DIR	5 4 LDX 3 EXT	6 5 LDX 3 IX2	5 4 LDX 2 IX1	4 3 LDX 1 IX	E 1110
F 1111	10 5 BRCLR7 3 BTB	7 5 BCLR7 2 BSC	4 3 BIH 2 REL	6 5 CLR 2 DIR	4 3 CLR 1 A	4 3 CLR 1 X	7 6 CLR 2 IX1	6 5 CLR 1 IX	2 WAIT 1 INH	2 2 TXA 1 INH		5 4 STX 2 DIR	6 5 STX 3 EXT	7 6 STX 3 IX2	6 5 STX 2 IX1	5 4 STX 1 IX	F 1111

Abbreviations for Address Modes

INH	Inherent
A	Accumulator
X	Index Register
IMM	Immediate
DIR	Direct
EXT	Extended
REL	Relative
BSC	Bit Set/Clear
BTB	Bit Test and Branch
IX	Indexed (No Offset)
IX1	Indexed, 1 Byte (8-Bit) Offset
IX2	Indexed, 2 Byte (16-Bit) Offset

LEGEND



APPENDIX I
MEMORY MAP

The Memory Map provides a quick reference as to the available bytes of addressable address spaces. Note that the first 128 bytes are relatively fixed. However, the number of remaining bytes and their function depends upon the actual device. See individual data sheet for specific memory map details.